

1.第一个C语言程序

平常随处可见的电脑想要运行起来，软件运行起来，需要我们人类告诉电脑如何运行，但是电脑又听不懂我们说的话该怎么办。这就创造了我们的语言: C语言，C++，JAVA，Python。但是我们语言这么多种我们为什么就偏要学C语言，C++呢？

那么C语言和C++有什么区别呢，具体来说C语言是C++的爹，因为C++是在C语言的基础上进行改进的。所以C语言能用的C++也能用，C++能用的C语言用不了，所以我们可以先学C语言再学C++。

下面一起来看一段代码:

```
1  #include<stdio.h> //include在英文是包含的意思，包含stdio.h里面的一些内容(C语言作者)
2
3  int main(){ //告诉电脑我们从这里开始运行
4
5      printf("Hello C语言"); //输出一句话 Hello C语言
6      printf("Hello C语言2"); //输出一句话 Hello C语言2
7      printf("Hello C语言3"); //输出一句话 Hello C语言3
8      printf("Hello C语言4"); //输出一句话 Hello C语言4
9      //printf("Hello C语言5");
10     return 0; //返回0
11 }
12
```

我们一句一句来解析到底干了什么事情

- `#include<stdio.h>`
include在英文是包含的意思，包含 `stdio.h` 里面的一些内容
这些内容是由创造C语言的人给我们的工具箱，就像下面的 `printf` 就是工具箱里面的一个工具
- `main`
他是一个 **函数**(后边会学) 他告诉电脑，从他这里开始运行代码，代码必须从上往下一句一句执行。(程序的入口)
- `{ }`
大括号里面写我们想要干的事情，包含内容。
- `printf`
可以输出中文，字母，数字等
- `return 0`
return是返回的意思，这句话就是告诉电脑，我做完了这件事得到了什么结果
- `" "`
双引号，是字符串的意思，里面的内容就是我们想要输出的东西
- `;`
这个标识相当于语文中的句号，表示我们说完了这句话，称作语句，只有计算机看得懂的合法操作才算是语句。
- `//`
注释，把一些不需要机器运行的句子给注释掉。

我们想要学好语言的第一件事就是要知道语言的规则是什么，如何写才能正确。

第一个提到的规则就是**语句的规则**：语句一定是需要计算机能看懂的东西，例如数字，字符，字符串。

```
1 10;  
2 "123123"; //在计算机中双引号引起来的叫做字符串  
3 "abc";  
4 'a'; //单引号引起来的叫做字符
```

练习:

1. 以下哪些是合法语句?

```
1 1;  
2 asd;  
3 "111";  
4 231;  
5 ();  
6 *;  
7 &;  
8 ^;  
9 '111';  
10 '&';  
11 "&^*%$";
```

2. 试验以下操作

- 1.如果写两个 `main` 函数会怎么样?
- 2.如果不写 `return` 会怎么样
- 3.如果写 `return 1`, `return 2` 会怎么样
- 4.如果不写 `#include<stdio.h>` 会怎么样
- 5.如果 `printf` 不写 `" "` 会怎么样

2.输入 输出

示例程序输出

```
1 #include<stdio.h>
2 int main(){
3     printf("我是大帅哥");
4     printf("我是大美女\n");
5     printf("我是小帅哥");printf(" 我是小美女");
6     return 0;
7 }
```

大家通过示例程序会发现最后输出如下

```
1 我是大帅哥我是大美女
2 我是小帅哥 我是小美女
```

以上代码体现了几个重点

- 3,4行输出代码写在不同行，但是最后输出并没有换行，说明我们主观上的换行不行
- 4行加入了 `\n` 结果输出换行了，说明 `\n` 是用来换行的
- 第5行写了两个语句并没有出现错误，说明一行可以写多句代码。

2.1输入

在平常使用电脑或者手机中，我们有很多方式可以输入，例如电脑的键盘，手机的触屏。那么需要输入的场景也是各种各样的，比如你在聊天的时候需要输入才能发送消息给你的朋友，那么输入到哪去？输入到聊天框去，那么也就是说聊天框变成了我们信息的载体，帮我们存储了这些信息。那么C语言也是一样的，当我们输入进去的东西如果没有东西给他存下来的话，那么这些信息就会消失。

输入以下步骤：

1. 使用输入工具 `scanf("")`
2. 选择输入的载体
3. 输入你的信息

但是目前我们好像并没有好的载体，所以就有了我们的**变量**。

变量就是在C语言中用来存储**信息的载体**，变量有各种各样的类型：整形，浮点数，字符等...

那么为什么变量需要这么多类型：举个例子我们在家里做菜的时候，有许多的调料：鸡精、味精、醋、酱油等，这些调料都是分别放在各自的盒子里，假如把放醋的放到酱油里，那么他们的味道就会混合，让我们的菜变得混乱，所以数据也是一样，为了防止数据混乱，我们创造了许多类型。同时大家可能会疑问：“老师不同类型我能够理解，为什么要创造长整型，整形呢？”这个问题很好解释：我们知道变量是信息的载体，那既然是载体那肯定需要空间，每个载体的空间都是需要向我们的计算机去申请的，倘若计算机空间不够这个变量就申请不了了，不仅如此申请成功的空间后，我们会给这个变量一个地址。我们都知道不同的物品大小需要的盒子大小就不同，所以对待不同大小的数据我们需要不同大小的变量，假如一串数字 1234567890123456789，和一连数字 123，我们就能知道 123所需要的空间肯定更小。

常见的变量类型有以下几种：

类型写法	类型含义	占位符
int	整形	%d
long long	长整形	%lld
float	单浮点数	%f
double	双浮点数	%lf
bool	布尔	无
char	字符	%c

既然我们有变量类型那肯定有变量名：所以代码一般这样写

```
1 int a; //合法语句
2 char b;
3 int c,d;
```

以上代码被称为**变量的声明**，就相当于书本的目录，告诉计算机从这一行开始我创造了一个什么类型的载体，但是这个载体内部一开始存的是什么我们不知道

变量类型被称为关键词，变量名被称为标识符。

取名也是有规则的：可以用小写字母，大写字母，数字和下划线(_)来命名。而且名称的第1个字符必须是字母或下划线，不能是数字。

有效的名称	无效的名称
wiggles	\$Z]**
cat2	2cat
Hot_Tub	Hot-Tub
_kcab	Tax rate

那么我们创造好了信息的载体，就开始把我们输入的信息放到载体里面去。

大家可以看到，我们上边表格是有一个栏叫做占位符，那么占位符又有什么用呢？我们先来看一段代码的例子。

```
1 #include<stdio.h>
2 int main(){
3     int a;int b;
4     scanf("%d",&a);
5     printf("%d",a);
6 }
```

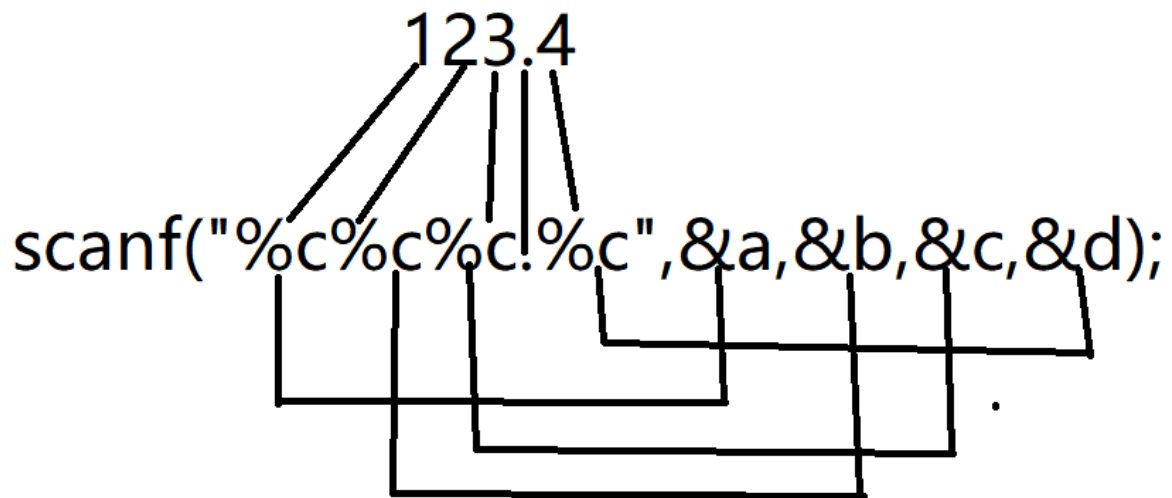
这段代码当我们输入什么数字，就会输出什么数字，但是当我们把 `%d` 去掉之后，连输入都没有输入程序就结束了。

那么这个占位符就是告诉计算机我在这里位置要输入什么数据，那么计算机就会给我们留位置输入，一个占位符对应一个变量。

举个例子：

```
1 char a,b,c,d;
2 scanf("%c%c%c.%c",&a,&b,&c,&d);
```

对于计算机来说的话，我们的scanf其实相当于一个原料分拣窗口，%c相当于一个接受字符类型的占位符，相当于分拣器的一个框框。当我们的计算机如果期望读到 "[一个字符] [一个字符] [一个字符] . [一个字符]" 按照这样子格式输入的数据。当读到123.4的时候他就会把这个读入拆分成"[1]" [2]" [3]" . [4]"，当每个框框都读取到足够的数据后，就会让后面的框框读取第一个分拣框把数据 '1'转交给了变量a，一直到第四个分拣框把数据'4'交给了变量d。那么转交给他的话就是需要我们的地址符号&



大家可以尝试一下不加上 & 会发生什么。导致这个的原因我举个例子大家就明白了。

就像爸爸妈妈让我们把饼干放盒子里面，那么饼干就一定在盒子里了吗？一定要我们去盒子所在的位置把饼干放进去才可以。

所以C语言也是一样 我们必须告诉计算机，这个变量的位置在哪里。

那么C语言作者给我们创造了一个符号 & ,&a就表示 变量a的位置。

所以全部都结合在一起就是

```
1 #include<stdio.h>
2
3 int main(){
4     int a;
5     long long b;
6     float c;
7     double d;
8     char e;
9     scanf("%d %lld %f %lf %c",&a,&b,&c,&d,&e);
10    printf("%d %lld %f %lf %c",a,b,c,d,e);//输出也是一样，但是请思考输出为什么不用加&
11    return 0; //返回0
12 }
```

输出和输入一样输出的占位符就是表示在这个位置输出，但是不需要加 & ，因为我们把东西存进去后，我们已经拥有了这一部分的记忆。

当然占位符不仅仅可以输出变量也可以输出常量例如如下：

```
1 #include<stdio.h>
2
3 int main(){
4     printf("%d %d %c",1,2,'a');
5     return 0; //返回0
6 }
```

注意:

对于浮点数来说是有保留几位小数的，c++很人性化给出了几个例子

```
1 printf("%.2f",a);就是保留两位小数输出
2 printf("%.3f",a);就是保留三位小数输出
```

2.2 赋值

变量我们说是信息的载体，所以对于变量来说我们除了用输入把信息放进去，我们还有一种方法就是赋值。

计算机赋值符号是 `=`。注意：不是等于

一般形式为 `变量 = 表达式;`

使用赋值会把原来存在载体中的信息覆盖掉。**=左边的叫做左值，右边的叫右值，常量不能当左值。**举例如下代码：

```
1 1 = 2; // 这样是错误的，因为 1 是常数所以不能当左值
2 int a = 1; //变量可以在声明的过程中就直接赋值
3 int c,d = 1,e = 2; //变量可以在声明的过程中就直接赋值
4 a = 2; //变量直接赋值为常量
5 int b = a; //变量也可以赋值为变量
```

但是大家请注意以下2点：

1. 由于赋值表达式 `=` 右边的表达式也可以是赋值表达式，因此下述表达形式
变量 = (变量 = 表达式); 这样子是成立的，其一般展开形式为：
变量 = 变量 = 变量 = = 表达式
例如 `a = b = c = 5`，它实际等价于 `c = 5; b = c; a = b;`
2. 在进行赋值运算时，如果赋值运算符两边的数据类型不同，系统会自动进行类型转换，即将赋值运算符右边的数据类型转换成左边的变量类型。当左边是整形而右边是浮点数时，将去掉小数部分并截取该整形对应的有效位数。例如：

```
1 int a = 123.567;
2 printf("%d",a); //输出123
```

示例程序

例1 输入两个整数 a 和 b，尝试交换 a，b 的值(使 a 的值等于 b，b 的值等于 a)

【分析】 交换两个变量值的方法很多，一般我们采用引入第三个变量的算法，两个变量交换，可以想象成一瓶酱油和一瓶醋交换，这时容易想到哪一个空瓶子过来：①将酱油倒入空瓶 ②将醋倒入酱油瓶 ③将原空瓶的酱油倒入醋瓶。程序如下：

```
1 #include<stdio.h>
2 int main(){
3     int a,b,c; //声明三个变量
4     scanf("%d %d",&a,&b);
5     c = a; a = b; b = c;
6     printf("%d %d",a,b);
7 }
```

例2 求三个整数的和

输入 a,b,c三个整数，输出它们的和。

【分析】 我们有两种选择，第一种选择定义四个整形变量 a,b,c,d。其中 a,b,c 用来存储输入，d 用来存储 `a + b + c` 的和最后输出 s，第二种就是直接按照整形输出 `a + b + c` 的值。

```

1  #include<stdio.h>
2  int main(){
3      int a,b,c; //声明三个变量
4      scanf("%d %d %d",&a,&b,&c);
5      int s = a + b + c;
6      printf("%d",a + b + c);
7      //printf("%d",s);
8  }

```

例3 小明买图书

已知小明有 n 元，他买了一本书，这本书原价为 m 元，现在打 8 折出售。求小明还剩多少钱(保留2位小数)。

【分析】 这个题主要考察格式化输出的方式，保留两位小数。

```

1  #include<stdio.h>
2  int main(){
3      double n = 0,m = 0,c = 0,d = 0;
4      scanf("%lf %lf",&n,&m);
5      c = m * 0.8;
6      d = n - c;
7      printf("%.2lf",d);
8      //printf("%d",s);
9  }

```

练习:

- 输入一个整形 a 以下哪个选项正确 ()
A. `scanf("%d",&a);` B. `scanf("%c",&a);` C. `scanf("%d",a);` D. `scanf("%f",a);`
- 输出一个双浮点数 a 以下哪个选项正确 ()
A. `printf("%d",a);` B. `printf("%c",&a);` C. `printf("%f",&a);` D. `printf("%lf",a);`
- 输出我是大傻蛋 以下哪个选项正确 ()
A. `printf("我是大傻蛋");` B. `printf(我是大傻蛋);` C. `printf("我是大傻蛋")`
- 请手写 分别输入整形 a，字符 b，单浮点数c 并且输出出来 (每输出一个换一行)。
- 请问以下哪些操作是合法的;

```

1  int a = 1;
2  a = -a;

```

```

1  char a = 'a';

```

```

1  char a = '1';

```

```

1  char a = '123';

```

```

1  char a = "1";

```

```

1  float a = 1;

```

```

1  double a = 0

```

```
1 int a;  
2 a = 1;
```

```
1 int a;  
2 scanf("%f",&a);
```

```
1 int a;  
2 scanf("%d",a);
```

```
1 int a = 2;  
2 scanf("%d",&a);
```

```
1 printf("%d",1);
```

```
1 int a = 1;  
2 printf("%d",&a);
```

```
1 printf("");
```

```
1 printf('%c','a');
```

```
1 printf("12314%d",1);
```

```
1 printf("%d",1,2,3);
```

```
1 printf("%d %d %d",1);
```

```
1 printf("123""123");
```

```
1 printf("123","123");
```

```
1 int a = 1,b,c,d = 1;
```

```
1 int a,int b,int c;
```

6. 下面的一个程序想要问一下你的意见有没有错误

```
1 #include<stdio  
2  
3 int main(){  
4  
5     int a,b,c = 2;  
6     printf("%d",a,b);  
7     printf("%d %d",c)  
8     return;  
9
```

7. 请问以下代码会输出什么?

```
1 printf("Baa Baa Black Sheep.");
```



```
1 int num;
2 num = 2;
3 num = 3;
4 printf("%d + %d = %d",num,num,num + num);
```

```
1 printf("123","1234");
```

```
1 printf("123123""123");
```

```
1 printf("123%d""123%d",4,5);
```

```
1 printf("%d%d",1,2);
```

```
1 int a = 100;
2 int b = a = 12 * 10;
```

8. 以下哪个选项能在电脑上输出() (多选)

```
1 *
2 ***
3 *****
4 ***
5 *
```

A. `printf(" *");printf(" ***");printf(" *****");printf(" ***");printf(" *");`

B.

`printf(" * \n");printf(" *** \n");printf(" ***** \n");printf(" *** \n");printf(" * \n");`

C. `printf(" * \n *** \n ***** \n *** \n * \n");`

9. 考虑以下程序

```
1 #include<stdio.h>
2 int main(){
3     int a,b;
4     a = 5;
5     b = 2;//第7行
6     b = a;//第8行
7     a = b;//第9行
8     printf("%d %d",b,a);
9     return 0;
10 }
11 //请问执行完第7, 8, 9行后程序的状态分别是什么 (a等于几, b等于几)
```

3.运算符

运算符有特别多，我们现在只讲几个常用的运算符，后面的学习运算符会越来越多

3.1算术运算符

用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称作取模运算,%)。

- 模运算符

求余的运算符 % 也称作模运算符，是一个双目运算符，俩个操作数都是整型数。a % b 的值就是 a 除以 b 的余数，5 % 2 = 1，其操作对象只能为整数。注意 % 的原理就是 a % b，为 a - a / b * b (a / b 为向下取整)。

- 除法运算符

C语言中的 / 和数学中的不太一样，数学中 1 / 2 = 0.5 但是C语言是分为整形，浮点数的，所以C语言中整形 / 整形 = 整形(向0取整)，浮点数 / 整形 = 浮点数，浮点数 / 浮点数 = 浮点数，整形 / 浮点数 = 浮点数

注意：取模只能整数对整数，不能其他类型！，并且取模也不能对0取模

```
1  #include<stdio.h>
2
3  int a,b,c;
4  double d;
5  int main(){
6      a = 10,b = 100,c = 10,d = 1.0;
7      printf("%d %d %d %d %d %d %lf\n",a * b,a + b,c - b,(a + b) * c,a + b * c,d / a);
8      return 0; //返回0
9  }
10 //输出 1000 110 0 1100 1100 0.1000000
```

当然C语言中也存在运算优先级，括号优先，乘法除法取模再次，加法减法最后。

3.2关系运算符

关系运算符一般用于比较运算。包括 大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=) 和 不等于(!=)。

他们都是二元运算符，也就是说有两个操作数，例如使用就是 a > b, a == b 等

对于这些操作数也是有结果的，也就是真和假，计算机在给出这些结果中以 1 代表真，0 代表假，但是反过来在判断一个值是真还是假的时候以 0 代表假，用 非0 代表真。

```
1  printf("%d %d %d %d",2 > 1,2 == 2,2 != 2,1 <= 2);
2  //最后会输出 1, 1, 0, 1
```

3.3逻辑运算符

逻辑运算符包括 与(&&)、或(||)、非(!)。

与运算符 和 或运算符均为双目运算符，具有左结合性，非运算符为单目运算符，具有右结合性。

- 与运算符：也就是 和 的意思，只有参与运算的两个都为真才为真，有一个为假就为假。

例如 5 > 0 && 4 > 2，由于 5 > 0 为真，4 > 2 为真，所以与出来的结果也为真。

- 或运算符：也就是 或者 的意思，只要参与运算的有一个为真就为真，两个都为假才为假。

例如 5 > 0 || 5 > 8，由于 5 > 0 为真，5 > 8 为假，所以或出来的结果也就为真。

- 非运算符：就是把假的变为真的，真的变为假的。

例如 !(5 > 0)，由于 5 > 0 为真，进行非运算后最终的结果为假。

逻辑运算符和其他运算符优先级的关系可表示如下：

按照与运算符的优先顺序可以得出：

`a > b && c > d` 等价于 `(a > b) && (c > d)`

`!b == c || d < a` 等价于 `((!b) == c) || (d < a)`

`a + b > c && x + y < b` 等价于 `((a + b) > c) && ((x + y) < b)`

3.4运算符的简写

在C语言中，有一些运算符可以简写，如下表：

简写	含义
<code>a += b</code>	<code>a = a + b</code>
<code>a -= b</code>	<code>a = a - b</code>
<code>a *= b</code>	<code>a = a * b</code>
<code>a /= b</code>	<code>a = a / b</code>
<code>a %= b</code>	<code>a = a % b</code>

运算符的优先级如下：

优先级	运算符	结合律
1	后缀运算符： <code>[]</code> <code>()</code> <code>·</code> <code>-></code> <code>++</code> <code>--</code> (类型名称){列表}	从左到右
2	一元运算符： <code>++</code> <code>--</code> <code>!</code> <code>~</code> <code>+</code> <code>-</code> <code>*</code> <code>&</code> <code>sizeof</code> <code>_Alignof</code>	从右到左
3	类型转换运算符： <code>(类型名称)</code>	从右到左
4	乘除法运算符： <code>*</code> <code>/</code> <code>%</code>	从左到右
5	加减法运算符： <code>+</code> <code>-</code>	从左到右
6	移位运算符： <code><<</code> <code>>></code>	从左到右
7	关系运算符： <code><<=</code> <code>>>=</code>	从左到右
8	相等运算符： <code>==</code> <code>!=</code>	从左到右
9	位运算符 AND： <code>&</code>	从左到右
10	位运算符 XOR： <code>^</code>	从左到右
11	位运算符 OR： <code> </code>	从左到右

练习：

- 下面的运算是真是假？
正整数 `>=` 负数
整数 `>=` 负数
`1 >= 2`
`2 >= 2`
`-2 >= -1`

-2 < -1
1 == 2
1 != 2
2 > 1 > 1
10 < 1 < 1
10 != 2 != 1
10 != 2 != 0
2 >= 1 >= 1
5 > 1 > 2 > 1 > 1 != 0
2 == 1 == 2 == 3 == 0

2. 给定3个整数 a, b, c。计算出表达式 (a + b) * c

3. 给定3个整数 a, b, c。计算出表达式 (a + b) / c

4. 给定被除数 a, 除数 b, 求出整数商和余数

5. 输入一个数字 a, 假设 a 是偶数则输出0, 否则输出 1

6. (3 >= 2) && (5 >= 3) 真还是假?

(3 >= 2) && (2 >= 3) 真还是假?

(1 >= 2) && (2 >= 3) 真还是假?

(3 >= 2) || (5 >= 3) 真还是假?

(3 >= 2) || (2 >= 3) 真还是假?

(1 >= 2) || (2 >= 3) 真还是假?

(3 >= 2) && (! (5 >= 3)) 真还是假?

(3 >= 2) && (! (2 >= 3)) 真还是假?

(1 >= 2) && (! (2 >= 3)) 真还是假?

(1 >= 2) || (! (2 >= 3)) 真还是假?

(1 > 2 > 3) && (1 < 2 < 3) 是真还是假?

4.计算机中的数据

4.1 原码反码补码

在我们使用以上运算符的时候我们有时候会得到错误的值，例如：

```
1 #include<stdio.h>
2 int main(){
3     int a = 1000000000,b = 100000000000;
4     printf("%d %d",a + b,a * b);
5 }
```

我们发现与我们的预期值不一样，为什么相乘得到了负数，相加变小了。

这个时候就要说到我们的数据存储，以及数据大小了。

给大家举个例子，我们现实中的盒子有大有小，大的装的东西比较多，小的装的东西比较小，如果我们用大的容器装很少的东西我们会觉得很浪费，如果用小容器装很大的东西又装不下，计算机也是一样，它能装的东西有限，所以C语言作者给每个变量都设置了一定的大小。

变量名	变量翻译	占用内存
int	整形	4个字节
long long	长整形	8个字节
float	单浮点数	4个字节
double	双浮点数	8个字节
bool	布尔	1个字节
char	字符	1个字节

那么到底什么是字节呢，字节就是最小的单位吗？其实计算机中最小的单位是位，1个字节(Byte) =8位(Bit)。什么是位？就是我们在数字中的个位十位，这种位。

在计算机中数据全都是跟二进制有关来存储的，那么我们先来讲讲二进制这些事。

首先要讲二进制的话，我们首先就要谈到我们的十进制数字也就是我们平常使用的个十百千万这种数字。十进制的数字每一位有十种状态，从0~9，一共有十个数字，十进制逢十进一。首先我们来看一个例子：

$12345 = 1 * 10^4 + 2 * 10^3 + 3 * 10^2 + 4 * 10^1 + 5 * 10^0$ （x^y称作x的y次幂，也就是 y 个 x 相乘的结果，x⁰是1）

一般来说我们对于数字从右往左分别称作第一位，第二位..... 那么12345的第一位就是 5，第二位就是 4 那么二进制就很简单，也就是每一位有两种状态，0~9，一共有两个数字，逢2进 1，我们来看一个例子：

$$101001 = 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 41$$

其实41 和 101001 的本质是一样的，只不过表示形式不一样，所以 $41_{10}/x$ 和 $101001_2/x$ 产生的答案是一样的。

二进制转换成十进制其实很简单，就按照我们上述方法即可，大家可能会对十进制转换成二进制比较难以接受。

方法：

我们可以先把值转换成二进制的第一位，再通过逢2进1的方法慢慢向更高位进。例如：

$$41 \rightarrow 00000(41) \rightarrow 0000(20)(1) \rightarrow 000(10)(0)(1) \rightarrow 00(5)(0)(0)(1) \rightarrow 0(2)(1)(0)(0)(1) \rightarrow (1)(0)(1)(0)(0)(1)$$

也就是第一位拿到数后对 2 取模，剩下的就是第一位的数，商就是需要进位的数，同样的第二位对 2 取模，剩下的就是第二位的数，商就是需要进位 以此类推我们最终可以得到最终答案。

那么这种方式也可以这么看：我们直接一步到位从最高开始，依次降低每一位。

二进制的运算其实也很简单就是逢二进一就好了举个例子：

$101011_2 + 1110_2$ ，和十进制一样，位数不够的可以补0，也就是说左边的式子可以变为 $101011_2 + 001110_2$ ，随后从第一位开始进行加法即可，但是我们要 注意一个点也就是，二进制下是逢二进一。

$$\begin{array}{r} 101011 \\ + 001110 \\ \hline 111001 \end{array}$$

二进制的乘法与十进制也是一样，只不过乘起来的数也是逢二进一。

$101011_2 * 1110_2$ ，和十进制一样，位数不够的可以补0，也就是说左边的式子可以变为 $101011_2 * 001110_2$

$$\begin{array}{r} 101011 \\ \times 001110 \\ \hline 000000 \\ 101011 \\ 101011 \\ 101011 \\ \hline 101011010 \end{array}$$

讲完二进制后，我们就开始讲与我们计算机有渊源的三个码：**原码、反码、补码**。

首先就是二进制为什么不能当计算机的码，最直接的原因就是，二进制无法直接表示负数，需要加上负号。

原码

原码是在二进制的基础上发明的，原码在二进制的基础上新增了一个叫做符号位，将二进制下的最高位变成了符号位，因为正号负号刚好两种状态，由于 $(-1)^0$ 为 1， $(-1)^1$ 为 -1，所以在原码中也是这么定义的最高位为 1 的时候表示这个数是负数，最高位为 0 的时候表示这个数为正数。

举个例子：

$$(10011)_{\text{原}} = -3, (01101)_{\text{原}} = 13。$$

那么我们十进制转换成原码有一个好方法，就是先可以把这个十进制的绝对值 (负数的绝对值为相反数0，正数不变) 用二进制先写出来，最后再加上符号位。

例如：

-15_{10} 的绝对值是 15_{10} ， $15_{10} = 1111_2$ ，因为 -15_{10} 是负数，所以符号位应该为1，最终答案就是 11111_2

不过原码是一个有缺陷的码，原码在做加法运算的时候，假如符号位不同，符号位如何改变我们无法通过加减直接得出，不过乘法运算是正确的。

例如：

$(1011)_{\text{原}} + (0111)_{\text{原}} = (10010)_{\text{原}}$ ，但是这明显是错误的 $(1011)_{\text{原}} = -3$ ， $(0111)_{\text{原}} = 7$ ， $10010_{\text{原}} = -2$ ，所以运算结果不正确

反码

为了解决原码这种运算错误的情况，我们发明了补码，反码是在原码的基础上改进的，反码的定义：：正数的反码与其原码相同，负数的反码是对正数逐位取反 (0变成1,1变成0)

举个例子：

$$(10011)_{\text{原}} = (11100)_{\text{反}}, (01111)_{\text{原}} = (01111)_{\text{反}}$$

那么我们最重要的问题就是，当原码转换成反码后运算是否正确。

例如：

$$(1011)_{\text{原}} + (0111)_{\text{原}}, (1011)_{\text{原}} = (1100)_{\text{反}}, (0111)_{\text{原}} = (0111)_{\text{反}}, (1100)_{\text{反}} + (0111)_{\text{反}} = (11011)_{\text{反}} = (10100)_{\text{原}} = -4, \text{答案正确}$$

但是像这种拥有符号位的码还是有一个缺陷比如: $(1000)_{\text{原}} = -0$ $(0000)_{\text{原}} = 0$ ，但是正0，负0是毫无意义的，所以我们就舍弃了带有符号位的码

补码

为了解决+0 -0 的问题，我们创造了新的码叫补码，补码是直接对二进制的基础上进行改变，补码的定义：除最高位为负数位，其他位正常。

举个例子：

$$10110_2 = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3 + 1 * 2^4$$

$$10110_{\text{补}} = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3 + 1 * -2^4$$

我们也可以从补码的定义看出一些性质：当补码最高位为1的时候一定为负数，否则为非负数。

假设现在有一个二进制是正数，那么原反补的符号位都为 0，因为其他位都没有改变，所以**正数的 原码 = 反码 = 补码**

假设现在有一个二进制是负数，原反补的符号位都为 1，我们已知 原码 = 反码符号位不变其他位取反，我们现在来讨论一下补码与他们的关系。

假设这是一个 5 位负数, $1ABCD_{\text{原}} = -(D * 2^0 + C * 2^1 + B * 2^2 + A * 2^3)$ 首先我们能确定补码最高位一定为 1, 也就是说补码现在是 $1EFGH = (H * 2^0 + G * 2^1 + F * 2^2 + E * 2^3 - 2^4) = -(D * 2^0 + C * 2^1 + B * 2^2 + A * 2^3) = (-D * 2^0 - C * 2^1 - B * 2^2 - A * 2^3)$, 这里需要用到我们二进制的性质了, $2^0 + 2^1 + 2^2 + \dots + 2^k = 2^{k+1} - 1$ 。

现在原式变成 $(H * 2^0 + G * 2^1 + F * 2^2 + E * 2^3 - 2^0 - 2^1 - 2^2 - 2^3 - 2^0) = (-D * 2^0 - C * 2^1 - B * 2^2 - A * 2^3)$

$(H + D - 1 - 1) * 2^0 + (G + C - 1) * 2^1 + (F + B - 1) * 2^2 + (E + A - 1) * 2^3 = 0$

上述一组答案就是 $H + D = 2, G + C = 1, F + B = 1, E + A = 1$ 。

所以我们很容易看出来, A 和 E 相反, B 和 F 相反, G 和 C 相反, $D = 2 - H$ 。

也就是说当 D 为 1 的时候 H 为 1, D 为 0 的时候 H 为 2, 也就是进位, 总体来说 H 是 D 的相反 + 1。

那么最终来说, 每一位都是相反最后一位需要 + 1, 所以补码 = 反码 + 1

最终补码解决了所有问题, 所以我们选用补码为机器码

总结

计算机做的运算都是以补码形式进行运算

正数: 原码 = 反码 = 补码

负数: 原码, 反码 = 原码符号位不变其他位取反, 补码 = 反码 + 1

推导

通过以上分析我们就可以知道, **每一个类型的变量都是由一定的数据范围以及内存的**, 我们可以通过每个变量类型的占用内存来算出他们的范围。

8个字节的: $-2^{63} \sim 2^{63} - 1$

4个字节的: $-2^{31} \sim 2^{31} - 1$

1个字节的: $-2^{15} \sim 2^{15} - 1$

注意: 做符号运算的时候最终结果的类型会选择字节数最高的那个, 例如整形和长整型运算, 最后答案的结果是长整型。

4.2ASCII

上面提到存储数据是使用二进制的，那么英文字母，奇奇怪怪的字符，中文都是怎么存的？

为了解决这个问题，人们发明了ASCII表，当我们使用字符的时候C语言自动发生翻译产生我们想要的东西。

ASCII 字符代码表 一																						
高四位 低四位		ASCII非打印控制字符										ASCII 打印字符										
		0000					0001					0010	0011	0100	0101	0110	0111					
		0					1					2	3	4	5	6	7					
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL 空	16	▶	^P	DLE 数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH 头标开始	17	◀	^Q	DC1 设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☹	^B	STX 正文开始	18	↕	^R	DC2 设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX 正文结束	19	!!	^S	DC3 设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT 传输结束	20	¶	^T	DC4 设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ 查询	21	¢	^U	NAK 反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK 确认	22	■	^V	SYN 同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	●	^G	BEL 震铃	23	↑	^W	ETB 传输块结束	39	'	55	7	71	G	87	w	103	g	119	w	
1000	8	8	☐	^H	BS 退格	24	↑	^X	CAN 取消	40	(56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB 水平制表符	25	↓	^Y	EM 媒体结束	41)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	☐	^J	LF 换行/新行	26	→	^Z	SUB 替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT 垂直制表符	27	←	^[ESC 转意	43	+	59	;	75	K	91	[107	k	123	{	
1100	C	12	♀	^L	FF 换页/新页	28	└	^[_	FS 文件分隔符	44	,	60	<	76	L	92	\	108	l	124		
1101	D	13	♪	^M	CR 回车	29	↔	^]	GS 组分隔符	45	-	61	=	77	M	93]	109	m	125	}	
1110	E	14	🎵	^N	SO 移出	30	▲	^G	RS 记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☐	^O	SI 移入	31	▼	^-	US 单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	*Back space

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入

我们只需要记住A-Z的ascii码是 65~90 ， a - z的ascii码是97-122就行

4.3强制转换

我们在第2章说了当我们左值的变量类型和右值的变量类型不同的时候，我们会把右边的变量类型变成左边的变量类型，做运算的时候会选择字节数较大的变量类型作为最终结果，那么假如我们就像要选择特定的变量类型该怎么办，这个时候就要用到我们的强制转换。

举例：

```
1 int a = 9000000000,b = 9000000000;
2 printf("%lld",a * b);
```

上述代码想要按照长整型输出最终答案，但是因为 a 是整形，b 也是整形，所以 a * b 最终结果也是 整形，我们想要按照长整型输出是不对的，所以我们需要把其中的 a 或者 b 转换成长整型，他们最终相乘的结果也是长整型，所以我们可以这么写：

```
1 int a = 9000000000,b = 9000000000;
2 printf("%lld",(long long)a * b); //写法1: (转换后的变量类型)变量名
3 printf("%lld",long long(a) * b); //写法2: 转换后的变量类型(变量名)
```

练习：

1. C语言中运行的机器码是原码还是补码还是反码？
2. 以下哪个选项会溢出（）（多选）
 - A. int a = 10000000000; B.char a = 256 C.long long a = 1000000000
 - D.long long a; int b = 10000000;int c = 1000000000; a = b * c;
 - E.long long a; long long b = 10000000;int c = 1000000000; a = b * c;

F. long long a; int b = 10000000; long long c = 1000000000; a = b * c;

3. $(123)_{10} = (\quad)_{\text{原}} = (\quad)_{\text{反}} = (\quad)_{\text{补}}$

4. $(-17)_{10} = (\quad)_{\text{原}} = (\quad)_{\text{反}} = (\quad)_{\text{补}}$

5. $(\quad)_{10} = (101110)_{\text{原}} = (\quad)_{\text{反}} = (\quad)_{\text{补}}$

6. $(\quad)_{10} = (000110)_{\text{原}} = (\quad)_{\text{反}} = (\quad)_{\text{补}}$

7. $(\quad)_{10} = (\quad)_{\text{原}} = (100110)_{\text{反}} = (\quad)_{\text{补}}$

8. $(\quad)_{10} = (\quad)_{\text{原}} = (011000)_{\text{反}} = (\quad)_{\text{补}}$

9. $(\quad)_{10} = (\quad)_{\text{原}} = (\quad)_{\text{反}} = (111001)_{\text{补}}$

10. $(\quad)_{10} = (\quad)_{\text{原}} = (\quad)_{\text{反}} = (011100)_{\text{补}}$

11. $10 \% 3 = (\quad)$, $11 \% 3 = (\quad)$, $12 \% 3 = (\quad)$, $-5 \% 3 = (\quad)$, $-10 \% 4 = (\quad)$, $-100 \% 13 = (\quad)$

12. 字符存储的形式是什么? 是利用什么存储的?

13. 以下代码会输出什么?

```
1 int a = 120;  
2 char b = (int)a;  
3 printf("%d", (int)b);
```

```
1 int a = 1200;  
2 char b = (int)a;  
3 printf("%d", (int)b);
```

```
1 int a = 65;  
2 printf("%c", a);
```

```
1 char a = 'A';  
2 printf("%d", a);
```

5.字符串

字符串其实在前几章用的多，`""` 引起的叫做字符串，用 `' '` 引起的叫做字符，字符串是由任意个字符组成的，可以是0个也可以是100个。

大家最不好理解的其实是 **空字符串**：双引号里面什么都没有的就是空字符串 `""`，有些同学会认为 `" "` 加了空格的是空字符串，其实空格也算一个字符。

还要讲的就是我们的 `\n` 到底是什么，其实在我们计算机中去表示换行这类是很难得一件事情，所以我们就发明了转义字符，就是说在一个字符前加上 `\` 就能改变他的意思。

转义字符	意义	ASCII码值（十进制）
<code>\a</code>	响铃(BEL)	007
<code>\b</code>	退格(BS)	008
<code>\f</code>	换页(FF)	012
<code>\n</code>	换行(LF)	010
<code>\r</code>	回车(CR)	013
<code>\t</code>	水平制表(HT)	009
<code>\v</code>	垂直制表(VT)	011
<code>\\</code>	反斜杠	092
<code>\?</code>	问号字符	063
<code>\'</code>	单引号字符	039
<code>\"</code>	双引号字符	034
<code>\0</code>	空字符(NULL)	000
<code>\ddd</code>	任意字符	三位八进制
<code>\xhh</code>	任意字符	二位十六进制

6.一维数组

6.1一维数组的定义

如果数据多了的话其实我们取名字是件很麻烦的事情，就像我们人类取名字就是件麻烦的事情，很多人会重名，倘若一个班中有两个孩子叫做小明，老师此时叫小明，两个小明都会回答，所以就会造成一些错乱。

在计算机中也是这样，我们无法声明两个相同名字的变量，倘若标识符相同，计算机就会导致错乱，在计算机中取名其实挺简单，比如字母和数字的组合，像a1, a2, a3.....这样就解决了起名问题，因为数字是无穷无尽的，不过假如说你现在需要100个整形变量，int a1, a2, a3, a4.....这是一件很困难的事情，所以我们就发明了数组。

一维数组的定义格式如下：

变量类型 数组名[常量表达式]

说明：①数组名的命名规则与变量的命名规则一样

②常量表达式表示数组元素的个数，可以是常量和符号常量，但不能是变量

[] 这个符号代表数组，可以写成 `int a[10]`，念作声明了一个大小为 10 的数组，数组里的每个元素都是整形，其中a是这个数组的名字。

创建了 10 个整形元素，他们的名字分别为 `a[0]`，`a[1]` `a[9]`，记住 `a[10]` 并没有被创建

6.2一维数组的引用

通过给出的数组名称和这个元素在数组中的位置编号(即下标)，程序可以引用这个数组中的任何一个元素

一维数组的引用格式：数组名[下标]。例如：

```
1  #include<stdio.h>
2
3  int a[10];
4  int main(){
5
6      scanf("%d %d %d %d",&a[0],&a[1],&a[2],&a[3]);
7      a[0] = 10;
8      printf("%d %d %d %d",a[0],a[1],a[2],a[3]);
9      return 0; //返回0
10 }
```

一维数组的特点：

1. 一维数组的内存是一起申请的，比如 `int a[10]`，那么就是会向计算机申请 40 字节大小的内存。
2. 一维数组里面的元素是连续的，也就是类似于他们这么多元素申请了一栋房，分别住在各自的隔壁
3. 一维数组的下标从 0 开始
4. C语言只能逐个引用数组的元素，而不能一次性引用整个数组
例如： `int a[100], b[100]`。 `a = b`，这样子是错误的。

6.3一维数组的初始化

初始化是一个C语言的编程术语，就是把变量赋为默认值，把控件设为默认状态，把没准备的准备好，也就是说我们创建数组的时候，数组里面每个元素的值我们并不知道，所以我们可以采用初始化将他们变为我们想要的一写值。

数组的初始化可以在定义时一并完成。格式：

变量类型 数组名[常量表达式] = {值1, 值2,}

例如：

`int a[5] = {1,2,3,4,5};` 这样写等价于 `a[0] = 1, a[1] = 2, a[2] = 3, a[3] = 4, a[4] = 5;`

说明：

- ①在初值列表中可以写出全部数组元素的值，也可以写出部分。例如：以下方式也可以对数组进行初始化
`int a[10] = {1,2,3,4,5};` 这样写等价于 `a[0] = 1, a[1] = 2, a[2] = 3, a[3] = 4, a[4] = 5, a[5] = 0, a[6] = 0..... a[9] = 0;`
也就是说我们也可以对一部分初始化，剩下的全都自动变为0
- ②对数组元素全部初始化为0，可以简写为 `{}` 例如：
`int a[10] = {};`

6.4数组越界

C语言规定，使用数组时，要注意：

①数组元素的下标值一定要为非负整数

②在定义元素个数的下标范围内使用

然而，在程序中把下标写成负数，大于数组元素个数的时候，程序是可以正常运行的

例如：

```
1 int a[10];
2 a[-3] = 5;
3 a[20] = 15;
4 a[10] = 20;
5 int k = a[30]
```

虽然这些语句是合法的，但是他们访问的元素所在的内存，我们并没有申请，所以在使用的时候是不合法的，这种现象叫做数组越界。

练习：

1. 一个大小为100的整形数组 如果我们想要使用的话，以下哪个选项合法() (多选)

A.a[0] B.a[-1] C.a[99] D.a[100]

2. 想要给大小为3的整形数组中每个位置都输入一个数，以下哪个选项正确()

A scanf("%d %d %d",&a[0],&a[1],&a[2])

B scanf("%d %d %d",&a[1],&a[2],&a[3])

C scanf("%d %d %d",a[0],a[1],a[2])

D scanf("%d %d %d",a[1],a[2],a[3])

3. char a[100], float b[10000], char c[1123] 分别怎么念

4. 想要把大小为3的整形数组中存的每个数都输出出来，以下哪个选项正确()

A printf("%d %d %d",&a[0],&a[1],&a[2])

B printf("%d %d %d",&a[1],&a[2],&a[3])

C printf("%d %d %d",a[0],a[1],a[2])

D printf("%d %d %d",a[1],a[2],a[3])

5. int a[10]数组里面的每个元素是什么？每个元素的属性是否一样？

6. 请用代码写出以下题目

◦ 输入5个整数(不超过整形范围)，输出 第2个输入的，第3个输入的。

◦ 输入5个整数(不超过整形范围)，将这5个整数按倒序输出出来

7. 请问以下操作是否合法

```
1 int a[10];
2 scanf("%d",&a[11]);
```

```
1 | int a[100];  
2 | scanf("%d %d %d",&a[12],&a[12],&a[99]);
```

```
1 | int a[10];  
2 | a[1] = 1;  
3 | a[2] = 2;  
4 | scanf("%d %d",&a[3],&a[2]);  
5 | printf("%d %d",a[2],a[3]);
```

```
1 | float a[10];  
2 | a[1] = 1;  
3 | a[2] = 2;  
4 | a[3] = a[4] = a[5] = 7;
```

7.顺序结构

顺序结构的程序设计是最简单的，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。

例1：输入一个三位数，要求把这个数的百位数和个位数进行对调，输出对调后的数

【分析】先求出自然数的个位，十位，百位，然后将个位和百位对调

程序如下：

```
1  #include<stdio.h>
2  int main(){
3      int m;
4      cin >> m;
5      int a = m / 100; //百位数
6      int b = (m / 10) % 10; //十位数
7      int c = (m % 10); //个位数
8      int n = c * 100 + b * 10 + a; //重新组合对调后的数
9      printf("%d",n);
10 }
```

运行结果：

输入：10 20 输出：85.6667

例2：已知某班有男同学 x 位，女同学 y 位，x 位男生平均分是 87 分，y 位女生的平均分是 85，问全体同学平均分是多少分？

【分析】男女生的人数需要用户输入，然后根据题意 $(x * 87 + y * 85) / (x + y)$ 求出全体同学的平均分。

程序如下：

```
1  #include<stdio.h>
2  int main(){
3      int x,y;
4      scanf("%d %d",&x,&y);
5      printf("%.4f",1.0 * (x * 87 + y * 85) / (x + y));
6  }
```

运行结果：

输入：10 20 输出：432

例3：歌手大奖赛上 6 名评委给一位参赛者打分，6 个人打分的平均分为 9.6 分，如果去掉一个最高分，这名参赛者的平均分为 9.4 分；如果去掉一个最低分，这名参赛者的平均分为 9.8 分；如果去掉一个最高分和一个最低分，这名参赛者的平均分是多少？

【分析】首先求出 6 名评委的总分，然后根据去掉最高分的总分和最低分的总分，求出最高分的分值和最低分的分值，最后总分减去最高分和最低分除以 4 即是答案。

程序如下：

```
1  #include<stdio.h>
2  int main(){
3      float sc_all = 6 * 9.6; //求6名评委的总分
4      float sc_high = 5 * 9.4; //求去掉最高分后的总分
5      float sc_low = 5 * 9.8; //求去掉最低分后的总分
6      float high = sc_all - sc_high //求最高分
7      float low = sc_all - sc_low; //求最低分
8      float ans = (sc_all - high - low) / 4; //求平均分
9      printf("%.2f\n",ans);
10 }
```

运行结果：9.60

8.程序的控制结构

8.1概述

程序由若干条语句组成，各语句按照顺序一条一条地执行，这种顺序结构是简洁的。但在现实世界中，在解决问题的过程中，不可避免地遇到需要进行选择或需要循环工作的情况。这时，程序执行的顺序需要发生变化，而非从前向后逐一执行。因此，程序中除了顺序结构以外，通常还有选择结构、循环结构以及转移机制。

C++为了支持这些控制结构，提供了丰富、灵活的控制语句。从结构化程序设计的观点看，所有程序都可用3种控制结构即顺序结构、选择结构和循环结构实现。C++在默认的情况下采取顺序结构，除非特别指明，计算机总是按语句顺序一条一条地执行。为使程序更清晰、更易调试与修改，并且不容易出错，结构化编程要尽量少用或不用goto等跳转语句。

选择类语句包括if语句和switch语句，用它们来解决实际应用中按不同的情况进行不同处理的问题。如根据学生的成绩，对学生做出不同的等第评价。if选择结构称为单分支选择结构，选择或忽略一个分支的操作。if-else选择结构称为双分支选择结构，在两个不同分支中选择。switch选择结构称为多分支（或多项）选择结构，以多种不同的情况选择多个不同的操作。

循环类语句包括for循环语句、while循环语句和do循环语句三种，用它们来解决实际应用中需要重复处理的问题。如当统计全班同学总分时，就需要重复地做加法，依次把每个人的分数累加起来。

if、else、switch、while、do和for等都是C++关键字。这些关键字是该语言保留的，用于实现C++控制结构的不同特性。关键字不能作为变量名等一些标识符。注意，将关键字while的拼写变为“While”是个语法错误，因为C++是区分大小写的语言。while、if和else等所有C++保留关键字只能包含小写字母。

8.2 if选择结构

C语言提供三种选择结构，即 if 选择结构、if-else 选择结构 和 switch 选择结构。

if语句(单分支机构)

格式 1:

```
1  if (条件表达式)
2      语句1;
```

功能: 如果条件表达式的值为真，则条件成立，语句 1 会被执行。否则，语句 1 就会被忽略，程序将按照顺序从整个选择结构之后的下一条语句继续执行。

说明: 格式中的“条件表达式”必须用圆括号括起来

例1: 读入一个整数 a，如果 a 为偶数在屏幕上输出 yes。

```
1  #include<stdio.h>
2  int main(){
3      int a;
4      scanf("%d",&a);
5      if(a % 2 == 0) printf("yes");
6      return 0;
7  }
```

注意: 关系运算符 == 用来表示该符号的左右两边是否相等，不要写成赋值号 =。

若题目改成 读入一个整数 a，若 a 为奇数在屏幕上输出 no 该怎么写？

例2: 读入一个整数，若这个数大于 1 并且小于 100，则输出 yes。

```
1  #include<stdio.h>
2  int main(){
3      int a;
4      scanf("%d",&a);
5      if((a > 1) && (a < 100)) printf("yes");
6      return 0;
7  }
```

注意: 此程序的条件表达式为 (a > 1) && (a < 100)，根据要求“条件表达式”必须用圆括号括起来，否则运行会出错。

格式2:

```
1  if(条件表达式){
2      语句1;
3      语句2;
4      ...
5  }
```

若条件成立时，要执行的操作由多个句子构成，我们必须把这些句子括在一对花括号{ }内，我们称这种形式为语句块或复合语句。

例3: 输入三个整数，按从大到小的顺序输出。

【分析】 输入的三个整数放在 a、b、c 中，设想让 a 为三数中最大数，怎么办呢？如果 a < b，那么让 a 和 b 的值交换，保证 a >= b；如果 a < c，那么让 a 和 c 的值交换，就保证了 a >= c；设想让 b 为第二大的数，c 为第三大的数，怎么做呢？如果 b < c，那么让 b 和 c 的值交换，就保证 b >= c，最后输出 a，b，c 的值。程序如下：

```

1  #include<stdio.h>
2  int main(){
3      int a,b,c,temp;
4      scanf("%d %d %d",&a,&b,&c);
5      if(a < b){
6          temp = a;
7          a = b;
8          b = temp;
9      }
10     if(a < c){
11         temp = a;
12         a = c;
13         c = temp;
14     }
15     if(b < c){
16         temp = b;
17         b = c;
18         c = temp;
19     }
20     printf("%d %d %d",a,b,c);
21 }

```

if-else语句(双分支结构)

if 单分支选择结构只在条件为 true 时采取操作，条件 false 时则忽略这个操作。利用 if-else 双分支选择结构则可以在条件为 true 时和条件为 false 时采取不同操作。

格式1:

```

1  if(条件表达式)
2      语句1;
3  else
4      语句2;

```

功能：如果(条件表达式)的值为“真”，即条件成立，则执行语句 1，执行完语句 1 后继续执行整个 if-else 语句的后继语句。如果(条件表达式)的值为“假”，即条件不成立，那么跳过语句 1 选择执行 语句 2，执行完语句 2 后继续执行整个 if-else 语句的后继语句。也就是说 if-else 语句总是根据(条件表达式)的结果，选择 语句 1 和 语句 2 中的一个执行，执行完以后，整个 if-else 就算执行完了。

程序设计风格提示：书写 if-else 语句时，if 和 else 要对齐，而分支的语句要缩进两格。

例4：输入温度 t 的值，判断是否适合晨练。(25 <= t <= 30)，适合晨练输出 ok，否则输出 no

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      int t;
4      scanf("%d",&t);
5      if((t >= 25) && (t <= 30))
6          printf("ok");
7      else
8          printf("no");
9  }

```

格式2:

```

1  if(条件表达式){
2      语句1;
3      语句2;
4      ...
5  }else{
6      语句1;
7      语句2;
8      ...
9  }

```

若分支语句由多个句子构成，我们必须把这些句子括在一对花括号 { } 内。

例5：乘坐飞机时，当乘客行李小于等于 20 公斤时，按每公斤 1.68 元收费，大于 20 公斤时，按每公斤 1.98 元收费，编程计算收费(保留两位小数)。

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      float w,s;
4      scanf("%f",&w);
5      if(w <= 20){
6          s = w * 1.68;
7          printf("%.2f\n",s);
8      }else{
9          s = w * 1.98;
10         printf("%.2f\n",s);
11     }
12     return 0;
13 }

```

if 语句允许嵌套，即语句 1 和 语句 2 还可以是 if 语句，当 if 语句嵌套时，约定 else 总是和最近一个 if 语句配对。

例6：

```

1  if(a > b)
2      if(b > c) y = a;
3      else y = c;

```

else 部分否定的是条件 $b > c$ ，即它与第二个 if 语句配对；若想要让 else 语句与第一个 if 语句配对，则要引入一个复合语句，将上述语句写成如下形式：

```

1  if(a > b){
2      if(b > c) y = a;
3  }else y = c;

```

例7：输入三个整数，输出其中最大的数。

【方法1】设 maxn 用于存放三个数中最大的数，输入的三个数存放在 a, b, c 中，那么如果 a 比 b 和 c 大，则最大数是 a，否则，如果 b 比 a 和 c 大，则最大数是 b，否则，最大数是 c。

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      float a,b,c,maxn;
4      scanf("%f %f %f",&a,&b,&c);
5      if(a > b && a > c) maxn = a;
6      else if(b > a && b > c) maxn = b;
7      else maxn = c;
8      printf("%f",maxn);
9  }

```

【方法2】设 maxn 用于存放三个数中最大的数，输入的三个数存放在 a、b、c 中，初值 maxn = a，即假设 a 为最大，那么如果 b > maxn，则此时的最大数应该是 b 即 maxn = b，如果 c > maxn，则最大数应该是 c，即 maxn = c。

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      float a,b,c,maxn;
4      scanf("%f %f %f",&a,&b,&c);
5      if(b > maxn) maxn = b;
6      if(c > maxn) maxn = c;
7      printf("%f",maxn);
8  }

```

if-else if语句

if-else 语句是很绝对的，只有两种情况，但是我们对于 if-else if 我们可以书写多种情况

格式1：

```

1  if(条件表达式1)
2      语句1;
3  else if(条件表达式2)
4      语句2;
5  else if(条件表达式3)
6      语句3;
7      .....

```

功能：如果(条件表达式 1)的值为“真”，即条件成立，则执行语句 1，执行完语句 1 后继续执行整个 if-else if 语句的后继语句。如果(条件表达式)的值为“假”，即条件不成立，那么跳过语句 1 选择判断(条件表达式 2)，如果为“真”，则条件成立，则执行语句 2，执行完语句 2 后继续执行整个 if-else if 语句的后继语句。如果(条件表达式 2)为假，那么就会跳过语句 2，去执行(条件表达式 3)，如果为真则执行语句 3, 也就是说 if-else if 语句会先按照条件表达式的先后顺序来一直判断是否为真，一旦确定一个条件表达式为真则执行对应的语句，后续的不再进行。

例子8：输入一个整数 a，假设 a 整除 6 输出 yes，假设 a 整除 3 但不整除 2 则输出 no，假设 a 整除 2 但不整除 3 则输出 all right。

【分析】整除就是余数为 0，你不能以商来判断

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      int a;
4      scanf("%d",&a);
5      if(a % 6 == 0)
6          printf("yes");
7      else if((a % 3 == 0) && (a % 2 != 0))
8          printf("no");
9      else if((a % 2 == 0) && (a % 3 != 0))
10         printf("all right");
11 }

```

格式2:

```

1  if(条件表达式1){
2      语句1;
3      语句2;
4      ....
5  }else if(条件表达式2){
6      语句3;
7      语句4;
8      ....
9  }else if(条件表达式3){
10     语句5;
11     语句6;
12     ....
13 }
14 .....

```

若分支语句由多个句子构成，我们必须把这些句子括在一对花括号 { } 内。

例9: 输入一个整数 a，假设 a 整除 6 输出 yes 和 a / 6，假设 a 整除 3 但不整除 2 则输出 no 和 a / 3，假设 a 整除 2 但不整除 3 则输出 all right 和 a / 2。

【分析】整除就是余数为 0，你不能以商来判断

代码如下:

```

1  #include<stdio.h>
2  int main(){
3      int a;
4      scanf("%d",&a);
5      if(a % 6 == 0){
6          printf("yes\n");
7          printf("%d",a / 6);
8      }else if((a % 3 == 0) && (a % 2 != 0)){
9          printf("no\n");
10         printf("%d",a / 3);
11      }else if((a % 2 == 0) && (a % 3 != 0)){
12         printf("all right\n");
13         printf("%d",a / 2);
14     }
15 }

```

if-else if-else

格式1:

```

1  if(条件表达式1)
2      语句1
3  else if(条件表达式2)
4      语句2
5      ....
6  else
7      语句3

```

功能：先判断条件语句1，假设为“真”则运行语句 1，接着继续执行整个 if-else if-else 的后继语句，否则运行条件表达 3，如果为“真”则运行语句 2，接着继续执行整个 if-else if-else 的后继语句...如果前面的条件表达式都不符合则运行语句 3，也就是说 if-else if-else 一定有一个语句会被运行，但是 if-else if 不一定。

例子10：输入一个整数 a，假设 a 整除 6 输出 yes，假设 a 整除 3 但不整除 2 则输出 no，假设 a 整除 2 但不整除 3 则输出 all right，否则输出 ok。

【分析】整除就是余数为 0，你不能以商来判断

代码如下：

```

1  #include<stdio.h>
2  int main(){
3      int a;
4      scanf("%d",&a);
5      if(a % 6 == 0)
6          printf("yes");
7      else if((a % 3 == 0) && (a % 2 != 0))
8          printf("no");
9      else if((a % 2 == 0) && (a % 3 != 0))
10         printf("all right")
11     else
12         printf("ok");
13 }

```

格式2:

```

1  if(条件表达式1){
2      语句1
3      语句2
4      ....
5  }else if(条件表达式2){
6      语句3
7      语句4
8      ....
9  }
10 ....
11 else{
12     语句5
13     语句6
14     ....
15 }

```

练习

1. 下面的运算是真是假？

1. 正整数 >= 负数
2. 整数 >= 负数
3. 1 >= 2
4. 2 >= 2
5. -2 >= -1
6. -2 < -1

- 7. $1 == 2$
- 8. $1 != 2$
- 9. $2 > 1 > 1$
- 10. $10 < 1 < 1$
- 11. $10 != 2 != 1$
- 12. $10 != 2 != 0$
- 13. $2 >= 1 >= 1$
- 14. $5 > 1 > 2 > 1 > 1 != 0$
- 15. $2 == 1 == 2 == 3 == 0$

2. 下面的这个分支有可能输出什么

```
1  if(a >= 100){
2      printf("我是大聪明");
3  }else if(a >= 110){
4      printf("我是小聪明");
5  }else if(a >= 80){
6      printf("小聪明是我");
7  }else{
8      printf("大聪明是我");
9  }
```

3. 请用代码形式写出：输入一个整数，若是奇数则输出YES，否则就输出NO

4. 请用代码形式写出：输入一个整数，若这个数能够整除 3就输出 aaa，不然如果能整除 2 就输出 cccc，否则输出 dddd

5. 请用代码形式写出：输入一个整数，若这个数能够整除3并且能够整除2就输出aaa，否则不输出

6. 以下代码合法吗？若合法会产生什么结果？

```
1  int a = 1;
2  if(a >= 1) printf("%d",1);
3  else printf("%d",2);
```

```
1  int a = 2,b = 1,c = 1;
2  if(a > b > c)
3      printf("aaa");
```

```
1  int a = 1;
2  if(a >= 1);
3  else;printf("%d",1);
```

```
1  int a = 1;
2  if(a >= 1); printf("%d",2);
3  else;printf("%d",1);
```

```
1  int a = 1,b = 2;
2  if(a != 1) printf("%d",b);
3  else printf("%d",a);
```



```
1 int a = 1,b = 2;
2 if(a != 1) printf("%d",b);
3 else if(a != 2) printf("%d",b);
4 else printf("%d",a);
```

```
1 int a = 1,b = 2;
2 if(a != 1){
3     printf("YES");
4     printf("%d",b);
5 }else if(b != 2){
6     printf("%d",b);
7 }else{
8     printf("NO");
9 }
```

```
1 int a = 105;
2 if(a % 3 == 0){
3     if(a % 3 == 0){
4         printf("YES\n");
5     }else if(a % 5 == 0){
6         printf("NO\n");
7     }else{
8         printf("AAA\n");
9     }
10    if(a % 5 == 0){
11        printf("YES\n");
12    }else if(a % 7 == 0){
13        printf("NO\n");
14    }else{
15        printf("III\n");
16    }
17    if(a % 5 == 0){
18        if(a % 7 == 0){
19            printf("YES\n");
20        }else{
21            printf("NO\n");
22        }
23    }
24 }
```

```
1 if(1 > 2 > 0){
2     printf("YES\n");
3 }else if(2 > 1 > 3 > 0){
4     printf("NO\n");
5 }else if(2 < 1 < 3 != 0){
6     printf("AAAA\n");
7 }else{
8     printf("DDDD\n");
9 }
```

```
1 if(1 > 2 > 0){
2     printf("YES");
3 }else{
4     printf("Asdsa");
5 }else{
6     printf("asddd");
7 }
```

```

1 | int a = 1;
2 | if(13){
3 |     int a = 2;
4 |     printf("%d",a);
5 | }

```

```

1 | int a = 1;
2 | {
3 |     int a = 3;
4 |     printf("%d",a);
5 | }

```

7. 对于以下代码请分析每一句话是什么意思，如果有运算请写出先运算什么，用什么结果进行下一步运算
例如 $1 > 2 > 3$ 先运算 $1 > 2$ ，再运算 $(1 > 2 \text{的结果}) > 3$

```

1 | if(a != 1 > 2){
2 |     printf("%d",2);
3 | }else if(a != 1 < 2){
4 |     printf("%c",65);
5 | }else{
6 |     printf("DDD!");
7 | }

```

```

1 | if(a * 2){
2 |     printf("YES\n");
3 |     if(a / 2) printf("YES2\n");
4 |     else if(a % 2) printf("YES3\n");
5 | }else if(a % 2 == 0){
6 |     printf("YES3\n");
7 |     if(a % 3 == 0){
8 |         printf("YES4\n");
9 |     }
10 |     if(a % 4 == 0){
11 |         printf("YES5\n");
12 |     }
13 | }

```

8. 请问以下代码是否一定输出YES

```

1 | 已知 a % 4 == 0
2 | if(a % 2 == 0){
3 |     printf("YES");
4 | }

```

```

1 | 已知 a > 1 <= 2 为真
2 | if(a > 1){
3 |     printf("YES");
4 | }

```

```

1 | 已知a > 1 <= 0 为真
2 | if(a > 1);
3 | else{
4 |     printf("YES");
5 | }

```

```
1 | 已知 1 > ((2 > a) > 3) 为真
2 | if(a < 2){
3 |     printf("YES");
4 | }
```

9. 请以代码形式写出以下描述（写出最简答案）

```
1 | 如果整形a能够整除3并且整形a能够整除5则输出YES
2 | 否则如果整形a能够整除3则输出NO
3 | 否则输出DDDD
```

```
1 | 如果整形a能够整除3并且能够整除5并且能够整除7并且能够整除11并且能够整除13并且能够整除39则输出
   | YES
```

8.3 switch语句

应用条件语句可以很方便地使程序实现分支，但是出现分支比较多时，虽然可以用嵌套的 if 语句来解决，但是程序结构会显得复杂，甚至凌乱。为方便实现多情况选择，C语言提供了一种 switch 开关语句

1. 语句格式

```
1  switch(表达式){
2      case 常量表达式1:
3          语句序列1;
4          break;
5      case 常量表达式2:
6          语句序列2;
7          break;
8          ...
9      case 常量表达式n:
10         语句表达式n;
11         break;
12     default:
13         语句表达式n + 1;
14 }
```

该语句可以使用一次或多次 case 标号，但只能使用一次 default 标号，或者省略整个 default 部分；多个 case 标号也允许使用在同一个和语句序列的前面；每个语句标号由保留字 case 和后面的常量表达式及冒号组成，每个常量表达式通常为字面常量，如常数或字符。

2. 语句执行过程

switch 语句执行过程分为以下 3 步描述。

①计算出 switch 后面圆括号内表达式的值，假定为 M，若它不是整形，系统将自动舍去其小数部分，只取其整数部分作为结果值。

②依次计算出每个 case 后常量表达式的值，假定它们为 M1、M2、...，同样，若它们的值不是整形，则自动转换为整形。

③让 M 依次同 M1、M2、... 进行比较，一旦遇到 M 与某个值相等，则就从对应标号的语句开始执行；在碰不到相等的情况下，若存在 default 子句，则就执行其冒号后面的语句序列，否则不执行任何操作；当执行到复合语句最后的花括号时就结束整个 switch 语句的执行。

在实际使用 switch 语句时，通常要求当执行完某个 case 后的一组语句序列后，就结束整个语句的执行，而不让它继续执行下一个 case 语句后面的语句序列，为此，可通过使用 break 语句（结束语句）来实现。该语句只有保留字 break，而没有其他任何成分。它是一条跳转语句，在 switch 中执行到它时，将结束该 switch 语句，系统接着向下执行其他语句。

在使用 switch 语句时，还应注意以下几点：

- ① case 语句后的各常量表达式的值不能相同，否则会出现错误码
- ②每个 case 或 default 后，可以包含多条语句，不需要使用 "{" 和 "}" 括起来。
- ③各 case 语句的先后顺序可以变动，这不会影响程序执行结果。
- ④default 语句可以省略，default 后面的语句末尾可以不必写 break。

程序设计风格提示：写 switch 语句时，switch (表达式) 单独一行，各 case 分支和 default 分支要缩进两格并对齐，分支处理语句要相对再缩进两格，以体现不同层次的结构。

3. 语句格式举例

(1) 左右两边的书写格式是等价的

<pre> 1 switch(a){ 2 case 1: x++; break; 3 case 2: y++; break; 4 case 3: z++; break; 5 default: printf("Error"); 6 } 7 8 9 10 </pre>	<pre> switch(a){ case 1: x++; break; case 2: y++; break; case 3: z++; break; default: printf("Error"); } </pre>
--	---

(2)

```

1  switch(ch){
2      case 'a':
3      case 'A':
4          d1 = (x + y) / 2;
5          d2 = x * y - 2;
6          break;
7      case 'b':
8      case 'B':
9          d1 = (a + b) / 2;
10         d2 = a * b - 2;
11         break;
12     default:
13         printf("input error!");
14 }

```

说明：1.每个 case 后面的语句可以写在冒号后的同一行或换到新行写。

2.<语句序列1> ... <语句序列 n + 1> 都时一组语句，有时可为空。如 (2)。

例1：根据从键盘上输入的代表星期几的数字，对应输出它的英文名字。

```

1  #include<stdio.h>
2  int main(){
3      int weekday;
4      scanf("%d",&weekday);
5      switch(weekday){
6          case 1: printf("Monday\n"); break;
7          case 2: printf("Tuesday\n"); break;
8          case 3: printf("Wednesday\n"); break;
9          case 4: printf("Thursday\n"); break;
10         case 5: printf("Friday\n"); break;
11         case 6: printf("Saturday\n"); break;
12         case 7: printf("Sunday\n"); break;
13         default: printf("input error\n");
14     }
15 }

```

例2：一个最简单的计算器支持四种运算。输入只有一行：两个参加运算的数和一个操作符(+, -, *, /)。输出运算表达式的结果。考虑下面两种情况：

(1) 如果出现除数为0的情况，则输出:Divided by zero!

(2) 如果出现无效的操作符(即不为+, -, *, /之一)，则输出：Invalid operator! 输入样例：

34 56 +

输出样例：

【分析】设 num1、num2 存放两个参加运算的操作数，op 存放操作符。

- ① 当op为“+”号时，实现加法操作。
- ② 当op为“-”号时，实现减法操作。
- ③ 当op为“*”号时，实现乘法操作。
- ④ 当op为“/”号时，判断b值，如果不为0,则实现除法操作，如果为0,则输出:Divided by zero!
- ⑤ 当op不是上面四种操作符时，输出：“Invalid operator!”

程序如下：

```
1  #include<stdio.h>
2  int main(){
3      float num1,num2;
4      char op;
5      scanf("%f %f %c",&num1,&num2,&op);
6      switch(op){
7          case '+': printf("%f",num1 + num2); break;
8          case '-': printf("%f",num1 - num2); break;
9          case '*': printf("%f",num1 * num2); break;
10         case '/':
11             if(num2 != 0) printf("%f",num1 / num2); break;
12             else printf("Divided by zero"); break;
13         default: printf("Invalid operator!");
14     }
15     return 0;
16 }
```

例3：期未来临了，班长小 Q 决定将剩余班费 x 元钱，用于购买若干支钢笔奖励给一些学习好、表现好的同学。已知商店里有三种钢笔，它们的单价分别为 6 元、5 元和 4 元。小 Q 想买尽量多的笔（鼓励尽量多的同学），同时他又不想有剩余钱。请你编一程序，帮小 Q 制订出一种买笔的方案。

【分析】对于以上的实际问题，要买尽量多的笔，易知都买 4 元的笔肯定可以买最多支笔。因此最多可买的笔为 $x/4$ 支。由于小 Q 要把钱用完，故我们可以按以下方法将钱用完：若买完 $x/4$ 支 4 元钱的笔，还剩 1 元，则 4 元钱的笔少买 1 支，换成一支 5 元笔即可；若买完 $x/4$ 支 4 元钱的笔，还剩 2 元，则 4 元钱的笔少买 1 支换成一支 6 元笔即可；若买完 $x/4$ 支 4 元钱的笔，还剩 3 元，则 4 元钱的笔少买 2 支，换成一支 5 元笔和一支 6 元笔即可。

从以上对买笔方案的调整，可以看出笔的数目都是 $x/4$ ，因此该方案的确为最优方案。

程序如下：

```
1  #include<stdio.h>
2  int main(){
3      int a,b,c,x,y;
4      scanf("%d",&x);
5      c = x / 4;
6      y = x % 4;
7      switch(y){
8          case 0: a = 0; b = 0; break;
9          case 1: a = 0; b = 1; c--; break;
10         case 2: a = 1; b = 0; c--; break;
11         case 3: a = 1; b = 1; c -= 2; break;
12     }
13     printf("%d %d %d",a,b,c);
14 }
```

练习:

1. 晶晶赴约会

晶晶的朋友贝贝约晶晶下周一起去看展览,但晶晶每周的1、3、5有课必须上课,请帮晶晶判断她能否接受贝贝的邀请,如果能输出YES;如果不能则输出NO注意YES和NO都是大写字母!

2. 骑车与走路

在清华校园里没有自行车,上课办事会很不方便。但实际上。并非去办任何事情都是骑车快,因为骑车总要找车、开锁、停车、锁车等,这要耽误一些时间。假设找到自行车,开锁并车上自行车的时间为27秒;停车锁车的时间为23秒;步行每秒行走1.2米,骑车每秒行走3.0米。请判断走不同的距离去办事,是骑车快还是走路快。如果骑车快,输出一行"Bike";如果走路快,输出一行"Walk";如果一样快,输出一行"All"。

3. 分段函数

编写程序,计算下列分段函数 $y=f(x)$ 的值。结果保留到小数点后三位。

$$y = -x + 2.5; \quad 0 \leq x < 5$$

$$y = 2 - 1.5(x-3)(x-3); \quad 5 \leq x < 10$$

$$y = x / 2 - 4.5; \quad 10 \leq x < 20$$

4. 计算邮资

根据邮件的重量和用户是否选择加急计算邮费。计算规则:重量在1000克以内(包括1000克),基本费8元。超过1000克的部分,每500克加收超重费4元,不足500克部分按500克计算;如果用户选择加急,多收5元。

5. 最大数输出

输入三个整数,数与数之间以一个空格分开。输出一个整数,即最大的整数。

6. 三角形判断

给定三个正整数,分别表示三条线段的长度,判断这三条线段能否构成一个三角形。如果能构成三角形,则输出"yes",否则输出"no"。

7. 判断闰年

判断某年是否是闰年。如果公元a年是闰年输出Y,否则输出N。

8. 点和正方形的关系

有一个正方形,四个角的坐标(x,y)分别是(1,-1),(1,1),(-1,-1),(-1,1),x是横轴,y是纵轴。写一个程序,判断一个给定的点是否在这个正方形内(包括正方形边界)。如果点在正方形内,则输出yes,否则输出no。

9. 简单计算器

一个最简单的计算器,支持+, -, *, /四种运算。仅需考虑输入输出为整数的情况,数据和运算结果不会超过int表示的范围。然而:

1. 如果出现除数为0的情况,则输出:Divided by zero!

2. 如果出现无效的操作符(即不为+, -, *, /之一),则输出Invalid operator!

6.3循环1

如果我们要使得一个变量 $a + 1 + 2 + \dots + 10000$ ，我们有两种办法，一种是自己一个一个加上去，一种是 $a + (1 + 10000) * 10000 / 2$ 。如果选择第一种方式的话，我们需要写10000个加法会很麻烦。所以我们C语言作者创造了循环，不断地做一件事情。

循环语句的模板 `for(;;){}`

第一个；前面写的是在循环开始前的准备活动，相当于一个语句(循环开始前执行，并且只执行一次)但是变量定义在里面和外面是有区别的

第二个；前面写循环进行的条件，例如 $i \leq 4$ ，表示的是变量 i 如果 ≤ 4 的话就继续进行循环，也可以不写(如果不写就相当于一直循环)(每次开始前执行)

第二个；后面写每次循环结束后执行的操作(也可以不写)

`for(int i = 0; i <= 4; i = i + 1)`这句话就表示 我们定义一个整形变量，循环进行的条件是整形变量 $i \leq 4$ ，每次循环结束后让 $i = i + 1$

举几个例子

```
1  for(int i = 0; i <= 4; i = i + 1){
2      printf("%d ",i);
3  }
4  第一次做的事情是先判断i是否小于等于4 然后输出i，此时i = 0，再给i加上1，此时i = 1
5  第二次做的事情是先判断i是否小于等于4 然后输出i，此时i = 1，再给i加上1，此时i = 2
6  第三次做的事情是先判断i是否小于等于4 然后输出i，此时i = 2，再给i加上1，此时i = 3
7  第四次做的事情是先判断i是否小于等于4 然后输出i，此时i = 3，再给i加上1，此时i = 4
8  第五次做的事情是先判断i是否小于等于4 然后输出i，此时i = 4，再给i加上1，此时i = 5
9  第六次做的事情是先判断i是否小于等于4 然后发现大于4，退出循环
10 所以一共循环了5次，输出0 1 2 3 4
```

```
1  int sum = 0;
2  for(int i = 0; i <= 4; i = i + 1){
3      sum = sum + 1;
4      printf("%d\n",sum);
5  }
6  第一次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 1，sum = 1
7  第二次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 2，sum = 2
8  第三次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 3，sum = 3
9  第四次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 4，sum = 4
10 第五次做的事情是先判断i是否小于等于5 然后给整形变量sum加上1然后输出sum，最后给i加上1，此时i = 5，sum = 5
11 第六次做的事情是先判断i是否小于等于5 然后发现i > 4 所以退出循环
```



```

1  int sum = 0;
2  for(int i = 0; i <= 4; i = i + 1){
3      sum = sum + 1;
4      printf("%d ",sum);
5  }
6  第一次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum, 最后给i加上1, 此时i = 1,
   sum = 1
7  第二次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum, 最后给i加上1, 此时i = 2,
   sum = 2
8  第三次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum, 最后给i加上1, 此时i = 3,
   sum = 3
9  第四次做的事情是先判断i是否小于等于4 然后给整形变量sum加上1然后输出sum, 最后给i加上1, 此时i = 4,
   sum = 4
10 第五次做的事情是先判断i是否小于等于4 发现大于4退出
11 最后输出 1 2 3 4 5
12 我们发现写在循环外边的变量不会被重新定义, 还会保留原来的值

```

```

1  int sum = 0;
2  for(int i = 0; i <= 2; i = i + 1){
3      int x = 1;
4      x = x + 1;
5      printf("%d ",x);
6  }
7  第一次做的事情先判断i是否小于等于2 然后定义一个整形变量x, 并且赋值为1 然后给x加上1, 输出x, 最后给i加上1
8  第二次做的事情先判断i是否小于等于2 然后定义一个整形变量x, 并且赋值为1 然后给x加上1, 输出x, 最后给i加上1
9  第三次做的事情先判断i是否小于等于2 然后定义一个整形变量x, 并且赋值为1 然后给x加上1, 输出x, 最后给i加上1
10 第四次做的事情先判断i是否小于等于2 然后发现i > 2所以退出循环
11 最后输出 2 2 2
12 我们发现写在循环里面的变量, 每次会重新定义, 所以进到下一次循环后, 上一次的变量就不存在了。

```

以上的循环语句我们明确知道会循环几次, 这叫做次数循环语句

还有一种循环语句叫条件循环语句 while(){}, while循环和for循环不同的是, 只需要写循环进行下去的条件

while(), while括号里面的就是条件执行语句, 例如 while(a >= 1) 的意思就是 如果 a >= 1的话循环就继续执行

那么 we 想和上面for循环一样的话 就得写 int i = 0;while(i <= 4) {i = i + 1};

举个例子

```

1  int i = 0;
2  while(i <= 4){
3      printf("%d ",i);
4      i = i + 1;
5  }
6  //以上代码答案会输出 0 1 2 3 4, 因为i = i + 1写在输出后边
7  int i = 0;
8  while(i <= 4){
9      i = i + 1;
10     printf("%d ",i);
11 }
12 //以上代码答案会输出 1 2 3 4 5, 因为i = i + 1写在输出前边

```

那么 we 想要把整形数组的 第一个~第五个位置都填上我们就要这样写

```

1  for(int i = 0; i < 5; i = i + 1){
2
3      scanf("%d",&a[i]);
4
5  }
6
7  int i = 0;
8  while(i < 5){
9      scanf("%d",&a[i]);
10 }

```

注意：如果不写大括号的话，循环语句只会执行 后边的第一句话

```

1  for(int i = 1; i <= 10; i = i + 1) printf("%d\n",1);
2  printf("%d\n",2);
3  //这样子会输出10个1, 1个2
4  for(int i = 1; i <= 10; i = i + 1) ;
5  printf("%d\n",1);
6  //这样子只会输出一个1，因为我们计算机是通过判断有没有；来看是否是一句话

```

练习：

1. 请用代码形式写出: 输入字符填上字符数组a的1~10个位置
2. 请用代码形式写出: 输入10个整数若是奇数则输出YES否则输出NO，每输出一行换一行
3. 请思考如何在C语言中判断是否为质数，请用代码形式写出
4. 给你1000个整数，从这1000个数中找到其中的最小值，最大值并且输出出来
5. 《庄子》中说到，“一尺之棰，日取其半，万世不竭”。第一天有一根长度为 a 的木棍，从第二天开始，每天都要将这根木棍锯掉一半（每次除 2，向下取整）。第几天的时候木棍的长度会变为 1？
6. 以下代码循环几次？转换成while循环怎么写或者转换成for循环怎么写；

```
1 | for(int i = 1; i <= 10; i = i + 1);
```

```
1 | for(int i = 0; i < 10; i = i + 1);
```

```
1 | for(int i = 1; i <= 9; i = i + 2);
```

```
1 | for(int i = 1; i <= 9; i = i * 2);
```

```
1 | for(int i = 1; i < 9; i = i + 1, i = i * 3);
```

```

1  int i = 1;
2  while(i <= 5){
3      i = i + 1;
4      i = i * 2;
5  }

```

7. 以下操作是否合法？若合法输出的结果又是什么？转换成while循环怎么写？

```

1  for(int i = 1,j = 1; i <= 10, j <= 11; i = i + 1, j = j + 1){
2      printf("%d\n",j);
3  }

```

```

1  for(int i = 1,j = 1; i <= 10 && j <= 11; i = i + 1, j = j + 1){
2      printf("%d\n",j);
3  }

```

```

1  for(int i = 1,j = 1; i <= 10 || j <= 11; i = i + 1, j = j + 1){
2      printf("%d\n",j);
3  }

```

```

1  int i = 1;
2  for(printf("Hello"); i <= 5,printf("LL"); i = i + 1);

```

```

1  int i = 1;
2  for(printf("Hello"); i <= 5; i = i + 1,printf("LLL"));

```

```

1  for(int i = 1,printf("LLLL"); i <= 2; i = i + 2);

```

```

1  int i = 1,j = 1,h = 1;
2  for(; i <= 100,j <= 100,h <= 2;){
3      h = h * 2;
4      printf("Hello");
5  }

```

```

1  for(int i = 1; i <= 5; i = i + 1);
2  printf("%d",i);

```

```

1  for(int i = 1; i <= 10; i = i + 1){
2      i = i + 1;
3      printf("%d\n",i);
4  }

```

8. 以下操作是否合法？若合法输出的结果又是什么？转换成for循环怎么写？

```

1  int i = 1;
2  while(i <= 5){
3      printf("%d",i);
4      i = i + 1;
5  }

```

```

1  int i = 1;
2  while(i <= 5)
3      printf("%d",i);
4      i = i + 1;

```

```
1 int i = 1,j = 1;
2 while(i <= 5,j <= 10){
3     printf("%d",i);
4     i = i + 1;
5     j = j + 1;
6 }
```

```
1 int i = 1;
2 while(i <= 5 && j <= 10){
3     printf("%d",i);
4     i = i + 1;
5     j = j + 1;
6 }
```

```
1 int i = 1;
2 while(i <= 5 || j <= 10){
3     printf("%d",i);
4     i = i + 1;
5     j = j + 1;
6 }
```

```
1 int i = 1;
2 while(i <= 5,i++){
3     printf("%d\n",i);
4 }
```

```
1 int i = 1;
2 while(i++,i <= 6){
3     printf("%d\n",i);
4 }
```

配套练习:

7.变量的作用域和生命周期

这个产生的区别最主要就发生在我们的{ }和一些循环，分支语句中。

首先{ }是包含的意思，包含在内部的意思，循环分支如果不写大括号，默认包含后面的第一句话

7.1作用域

作用域，程序设计概念，通常来说，一段代码中所用到的名字并不总是有效/可用的而限定这个名字的可用性的代码范围就是这个名字的作用域

我们可以试一下以下代码

```
1 #include <stdio.h>
2
3 int main(){
4     {
5         int num = 0;
6     }
7     printf("%d\n",num);
8 }
```

我可以发现爆出了错误，如果我们改成以下代码我们就发现代码能够运行。

```

1  #include <stdio.h>
2
3  int main(){
4      {
5          int num = 0;
6          printf("%d\n",num);
7      }
8  }

```

所以我们可以把作用域抽象成这些名字可以在哪个地方使用。
大括号内部的只能在这个大括号内部里面使用

我们称写在大括号里买的变量为局部变量

写在大括号外边的就是全局变量

局部变量的作用域是变量所在的局部范围，全局变量的作用域就是整个工程

同样的，因为我们的循环，分支语句全是有大括号的，所以在这里面定义的变量，只能在这里面使用

7.2生命周期

变量的生命周期指的是变量的创建到变量的销毁之间的一个时间段

1. 局部变量的生命周期是：进入作用域生命周期开始，出作用域生命周期结束。
通过作用域的学习，我们发现不管是循环语句，判断语句里面写的变量只能在他们内部使用。
所以我们出了他们内部，我们再也不会用他们的这些变量，所以系统会判断他为垃圾，自动销毁回收了
2. 全局变量的生命周期是：整个程序的生命周期。只有总程序结束了才会销毁

练习:

1. 以下哪些操作是合法的，并且说出每个变量的生命周期和作用域

```

1  #include<stdio.h>
2  int b = 1;
3  int main(){
4      int a = 1;
5      {
6          printf("%d",b);
7          printf("%d",a);
8      }
9  }

```

```

1  #include<stdio.h>
2  int main(){
3      {
4          int a = 1;
5          printf("%d",a);
6      }
7      printf("%d",1);
8  }

```

```

1  #include<stdio.h>
2  int main(){
3      {
4          int a = 1;
5      }
6      printf("%d",a);
7  }

```

```

1 #include<stdio.h>
2 int main(){
3     if(1){
4         int a = 1;
5     }
6     printf("%d",a);
7 }

```

```

1 #include<stdio.h>
2 int main(){
3     for(int i = 1; i <= 10; i = i + 1){
4         int a = 1;
5     }
6     printf("%d",i);
7     printf("%d",a);
8 }

```

```

1 #include<stdio.h>
2 int main(){
3     {
4         int a = 1;
5     }
6     for(int i = 1; i <= 10; i = i + 1){
7         a = a + 1;
8     }
9     printf("%d",a);
10 }

```

```

1 #include<stdio.h>
2 int main(){
3     for(int i = 1; i <= 10; i = i + 1){
4         int a = 1;
5         {
6             a = a + 1;
7         }
8         printf("%d",a);
9     }
10 }

```

8.循环中的关键词

有两个我们常用的关键词一个是continue，一个是break

1. continue是跳过的意思，也就是说跳过此次循环的意思。举个例子

```

1 for(int i = 1; i <= n; i = i + 1){
2     sum += 1;
3     continue;
4     sum += 2;
5 }
6 //这个代码，sum += 2不会执行，因为每次遇到了continue就跳过了此次循环

```

2. break是结束的意思，也就是说结束循环。举个例子

```

1  for(int i = 1; i <= n; i = i + 1){
2      sum += 1;
3      break;
4      sum += 2;
5  }
6  //这个代码，sum += 1只会执行一次，因为遇到了break，所有循环结束

```

练习:

1. 以下代码会输出什么?

```

1  for(int i = 1; i <= 5; i = i + 1){
2      sum += 1;
3      break;
4      sum += 2;
5  }
6  printf("%d",sum);

```

2. 以下代码会输出什么?

```

1  for(int i = 1; i <= 5; i = i + 1){
2      sum += 1;
3      continue;
4      sum += 2;
5  }
6  printf("%d",sum);

```

3. 输入一个整数，如果这个整数不是1，继续输入，重复这个动作。

9.分支3

在分支2的时候我们学了 &&(与运算符)，||(或运算符)。那么这节课我们要讲一下基本逻辑。

首先 &&和||的优先级是一样的，倘若没有括号我们是从前往后运算。

请大家回答以下 运算结果答案是否相同?

```

1  (a >= 0 && b >= 0) && c >= 0
2  a >= 0 && b >= 0 && c >= 0

```

```

1  a >= 0 && (b >= 0 && c >= 0)
2  a >= 0 && b >= 0 && c >= 0

```

```

1  a >= 0 || b >= 0 || c >= 0
2  (a >= 0 || b >= 0) || c >= 0

```

```

1  a >= 0 || b >= 0 || c >= 0
2  a >= 0 || (b >= 0 || c >= 0)

```

```

1  a >= 0 && (b >= 0 || c >= 0)
2  a >= 0 && b >= 0 || c >= 0

```

```

1  (a >= 0 && b >= 0) || c >= 0
2  a >= 0 && b >= 0 || c >= 0

```

```
1 | a >= 0 && b >= 0 || c >= 0
2 | c >= 0 || a >= 0 && b >= 0
```

练习:

- 当 $a = 3, b = 4, c = 5$ 判断以下表达式是否成立? (乘除运算优先级最高, 加减其次, 最后是关系运算符)
 - $a < b || a > c || a > b$
 - $a > c || b > a \&\& c > b$
 - $b - a == c - b$
 - $a * b - c > a * c - b || a * b + b * c == b * b * (c - a)$
- 当 $a = 1, b = 0, c = 1$, 判断以下表达式是否成立?
 - $!a || !b$ (先运算 !)
 - $(a \&\& !a) || (b || !b)$
 - $a \&\& b \&\& c || !a || !c$
 - $a \&\& (b \&\& c || a \&\& c)$
 - $!b \&\& (c \&\& (a \&\& (!c || (!b || (!a)))))$

10.运算符的使用

之前讲过 $+, -, *, /, \%$, 他们都是有二个操作数的, 什么叫二个操作数呢?

例如 $a + b$, a 是一个数字, b 也是一个数字, 所以叫做有二个操作数。

那么有二个操作数的被称作二元操作符。今天着重要讲的是, 一元操作符和三元操作符。

顾名思义, 就是一个操作数的和三个操作数的, 因为现实数学中用不到, 所以以下知识请不要在数学课上使用!

10.1一元操作符

一元操作符有很多, 下面介绍几个常用的

- ++ (称作自增, 通过自增可以使变量在自身的基础上加1)

使用方法为, $++a$, $a++$, 那么 $++a$ 和 $a++$ 有什么区别呢?

咱们可以通过两段代码来看出区别

```
1 | int a = 1;
2 | int b = ++a;
3 | printf("%d\n", b);
4 | b = a++;
5 | printf("%d\n", b);
6 | //我们发现这两段代码的输出都是2, 这下就可以看出来, ++a在使用前立刻会增加1然后再使用, a++是在使用后立刻会增加1。
```

- (称作自减, 通过自减可以使变量在自身的基础上减1)

使用方法为, $--a$, $a--$, 那么 $--a$ 和 $a--$ 有什么区别呢?

咱们可以通过两段代码来看出区别

```
1 | int a = 1;
2 | int b = --a;
3 | printf("%d\n", b);
4 | b = a--;
5 | printf("%d\n", b);
6 | //我们发现这两段代码的输出都是0, 这下就可以看出来, --a在使用前立刻会减少1然后再使用, a--是在使用后立刻会减少1。
```

- ! (称作否, 因为计算中有二种状态, 一种为真一种为假, 所以可以用它来表示相反的状态)

5. >> (称作右移，它也是对于二进制来说的，把二进制整体往右边移一位，左边补充和首位一样的数字)

- 1
- 举个例子，-4的二进制补码可以是 1100，1100>>1 = 1110 = -2，我们把最左边的那一位补上了1，如果是0的话答案就等于6了
- 2
- 4的二进制补码是0100，0100>>1 = 0010 = 2，我们把最左边的那一位补上了0，如果是1的话那就等于-2了。整体整除2。
- 3
- 大家还记得，我们的C语言中的 / 是整除的意思，向0取整，这就可以通过右移来看，因为计算机的运算其实最后都会转换成左移右移
- 4
- 例如3的二进制补码可以是 011，011>>1 = 001，最右边那一位是1，但是右移后没有他的位置了，所以直接没了，所以最后答案等于1，相当于向下取整
- 5
- 例如-3的二进制补码可以是101，101>>1 = 110，最右边那一位是1，但是右移后也没有他的位置了，所以直接没了，所以最后答案等于-2，相当于向下取整
- 6
- 此时有同学就有疑问了，负数整除不是向上取整么。
- 7
- 对，因为我们现实中为了保证 被除数 = 除数 * 商 + 余数，所以我们才保证向上取整

其实大家会的差不多了，不过还有更方便的写法现在交给大家比如：大家一般这么写

1

int a = 10;

2

int b = 2;

3

b = b + a;

4

b = b - a;

5

b = b ^ a;

6

b = b | a;

7

b = b & a;

8

b = b / a;

9

b = b % a;

10

//但是我们也可以如下这么写

11

b += a;//这句话同样表示给变量b加上a，

12

b -= a;

13

b ^= a;

14

b |= a;

15

b &= a;

16

b /= a;

17

b %= a;

优先级	运算符	结合律
1	后缀运算符：[] () · -> ++ --(类型名称){列表}	从左到右
2	一元运算符：++ -- ! ~ + - * & sizeof_Alignof	从右到左
3	类型转换运算符：(类型名称)	从右到左
4	乘除法运算符：* / %	从左到右
5	加减法运算符：+ -	从左到右
6	移位运算符：<< >>	从左到右
7	关系运算符：<<= >>=	从左到右
8	相等运算符：== !=	从左到右
9	位运算符 AND：&	从左到右
10	位运算符 XOR：^	从左到右
11	位运算符 OR：	从左到右

练习:

1. 思考以下代码会输出什么

```
1 | for(int i = 1; i <= 5; i++) printf("%d\n",i);
```

2. 思考以下代码会输出什么

```
1 | for(int i = 1; i <= 5; ++i) printf("%d\n",i);
```

3. ! 10000000, ! -100000000, ! 9999999999999999 分别输出什么?

4. 整形类型下~1, ~15, ~-5分别输出什么

5. ~(1000111)补, ~(0111111)补, ~(1111111)补分别输出什么

6. 123456 第1位, 第2位, 第3位, 第4位, 第5位, 第6位如何取? 用代码形式写出来

7. (101101)补 & (1110111)补, 15 & 14, 20 | 15, 0 | 100, 0 & 100, 0 ^ 100, 100 ^ 100, 100 ^ 100 ^ 100, 100 & 0 | 100 分别是多少

8. (2 != 1) ? 1 : 2, (5 > 2) ? 3 : 2, (1 + 2 != 3 > 4) ? 1 : 2。答案分别是什么

9. 10 >> 1, 15 >> 1, 10 << 2, 100 << 2, -100 / 2, -10 % 2, 10 % 2 分别输出什么(/是C语言中的整除)

10. 对于整形变量X, Y, 写出与判断以下性质对应的表达式

- (1) x是否为偶数
- (2) x是否为4位整数
- (3) 得到X在二进制形式下的 第3位
- (4) 判断X在二进制形式下的 第4位是否为1
- (4) 得到X, Y在二进制下的第 3 位的异或值

11. 编写代码实现: 求一个整数存储在内存中的二进制(补码)中1的个数

12. 不能创建临时变量(第三个变量), 实现两个数的交换。

11. 字符数组, 以及数组的构造

为什么要把字符数组单独拿出来呢, 因为在外国人眼里中文其实是不好表示的, 我们可以看我们之前的ascii码中并没有映射中文, 所以美国人用几个ascii码来表示一个中文, 例如他用32位来表示一个中文, 但是一个char是8位, 所以需要4个char才可以表示一个, 但是我们分开来存的时候, 如何把中文映射出来呢? 这个时候就需要我们的字符串输入和字符串输出登场了

```
1 | char a[10]; scanf("%s",&a);printf("%s",a);
```

由于字符只能存一个, 所以想存多个字符的话, 可以使用字符数组。那么输入的字符就一个个分别存在a[0],a[1],....

同学们可能有一些疑问：

- 为什么字符数组输出名字就能输出所有值？
- 字符数组怎么判定输出结束的？

11.1内存，地址的概念

每个计算机的内存是有限的，所以每个变量，每个数组，每句代码都会占用一定的空间，并且都会有地址，那么空间和地址到底是怎么分配的呢？

那么，我们假设我们的空间和我们的商品房一样

 image-20221028135708259

假设我们定义了一个整型变量a，一个整型变量b，那么他们两个会在这个空间里面找到一块地方放下。

 image-20221028135934903

那么数组的空间是怎么存储的呢？众所周知，数组是很多相同的变量组成的，所以他们的空间是连续的。例如我定义了 int a[5];

 image-20221028140529065

11.2为什么字符数组输出名字就能输出所有值，而整形数组输出的是地址？

首先变量的名字，代表了它的内容，但是数组包含了太多内容，例如整形数组，包含了好多整数，如果把他们直接输出的话，连在一起就产生了歧义，所以C语言作者让数组的名字去代表数组的地址。

然后因为C语言中没有专门处理字符串的东西，所以就选择了字符数组去处理，因为字符串是一整个可以连在一起的，拼接在一起的，但是像整数，浮点数，如果拼接在一起的话他们的意思就会改变，并且再分开也不知道如何分开了。所以C语言干脆就把字符数组单独做一个处理，输出字符串。

大家思考一个问题：如果我定义了一个变量a,他的首地址是0，他的末尾地址是4，那么我们知道 1 2 3 有任何意义吗？

答案是没有任何意义，因为我们无法通过 1 2 3推导出任何信息。所以C语言干脆就把 地址中的 + 1 - 1的意思给改变了

他把意思变成了: 以当前数据大小为单位进行加减

```
1 | int a[2];  
2 | printf("%d %d %d %d\n",&a,&a + 1,&a[0],&a[0] + 1); //大家觉得有什么区别呢？
```

最后我们发现，数组中名字代表的是一整个数组，而不是数组的元素。

11.3内存进阶内容

```
1 | int a[2];  
2 | printf("%d %d %d %d",a,&a,a + 1,&a + 1); //大家觉得他们之间有什么关联？
```

提示1：首先大家思考，a是什么，a[0]又是什么？

最后我们发现a也代表了数组第一个元素的地址，而不是数组的首地址(虽然他们的值相同，但是意思不同)

总结：

1. 定义一个数组a，a代表的是整个数组，而不是数组的元素
2. 定义一个数组a，a也可以代表数组第一个元素的首地址
3. 地址中加减的单位变成了，以我当前元素的数据大小为基础单位
4. 字符数组可以用名字直接输出字符串，其他不行

11.4字符数组如何判定结束

字符数组是用 '\0'来判定结束的

```
1 char a[10];
2 假设a[0] = 'A',a[1] = 'B',a[2] = 'C',a[3] = 'd',a[4] = '\0',a[5] = 'a';
3 printf("%s",a); 他会输出 ABCd
```

```
1 那么输入的时候
2 scanf("%s",a);
3 他会自动把输入的最后一个字符的后一个位置改成 '\0'，所以我们输出的时候会自动会停止
```

同时从这一块我们也能发现，如果给整形数组，浮点型数组也按照顺序输出的话，怎么样停止呢？是用0表示停止还是-1表示停止呢？一看就不合理

11.5数组的构造

首先**字符数组的构造**我们已经会了，当我的字符数组等于一个字符串的时候，他自动会把字符串拆分字符分别存进去

```
1 char a[10] = "123";
2 printf("%c %c %c",a[0],a[1],a[2]); //输出 1 2 3
```

整形数组和浮点数数组因为我们数字需要隔开，所以我们可以使用大括号包含我们想要构造的数字，然后中间用逗号隔开

例如

```
1 int a[10] = {0,1,2}; //默认从a[0]开始给，没有给到的赋值为0
```

浮点数数组也是一样

```
1 float a[10] = {0,1,2}; //默认从a[0]开始给，没有给到的赋值为0
```

练习:

1. 如何在定义的时候把大小为100的整形数组里的元素全都变成0
2. float a[10]; printf("%d %d %d %d %d %d",a,a + 1,&a,&a + 1,&a[0],&a[0] + 1); (假设数组a的首地址为0，请写出预期输出值)
3. int a = 2; printf("%d %d %d",a,&a,&a + 2); (假设a的首地址为0，请写出预期输出值)
4. int a[10] = {1,2,3}; printf("%d %d %d",a[0],a + 8,&a[8]); (假设a的首地址为0，请写出预期输出值)
5. 以下操作合法吗？若合法输出什么答案

```
1 scanf("%s",&a); //输入1234
2 printf("%s",a);
```

```
1 scanf("%s",a); //输入1234
2 printf("%s",a);
```

```
1 scanf("%s",&a); //输入1234
2 printf("%s",&a);
```

```
1 scanf("%s",a); //输入1234
2 printf("%s",&a);
```

```
1 scanf("%s",a); //输入1234
2 printf("%s",&a + 1);
```

```
1 char a[10] = {1,2,3,4,5,'\0'};
2 printf("%s",a + 1);
```

```
1 char a[10] = {1,2,3,4,5,'\0'};
2 scanf("%s",a + 1); //输出12345
3 printf("%s",a + 1);
```

```
1 char a[10] = {1,2,3,4,5,'\0'};
2 scanf("%s",a + 1); //输出1234
3 printf("%s",a + 1);
```

```
1 char a[10] = {1,2,3,4,5,'\0'};
2 a[2] = '\0';
3 scanf("%s",a + 1); //输出12345
4 printf("%s",a + 1);
```

```
1 int a[10] = {0,1,2,3};
2 scanf("%d",a + 2);
3 printf("%d",a[0]);
```

12.二维数组，循环2

二维数组顾名思义: 数组里面套一个数组

image-20220924173429317

读作: 大小为3的数组a里面的每个元素是大小为8的整形数组，也可以读作3行9列的二维数组a

所以我们数组a里面的元素不是整形而是数组，数组里面的数组的元素才是整形

就好比学校的元素是班级，班级里面的元素才是人

那么我想要使用我们的二维数组其实是一样 a[][]

我们想要输入a数组里第一个元素里的第一个元素 scanf("%d",&a[0][0]);

如果数组大小是 int a[100][100];单重循环来输入也是件麻烦事，所以我们会使用双重循环

双重循环:顾名思义 循环里面套循环

我们想要给数组int a[100][100]全部填上我们有两种写法

1. 先把每一行给填上

```
1 for(int i = 0; i <= 99; i++){
2     for(int j = 0; j <= 99; j++){
3         scanf("%d",&a[i][j]);
4     }
5 }
```

2. 先把每一列给填上

```

1  for(int i = 0; i <= 99; i++){
2      for(int j = 0; j <= 99; j++){
3          scanf("%d",&a[j][i]);
4      }
5  }

```

练习

1. 请用代码形式输出100以内的质数 (通过判断是否为质数联想)
2. 请用代码形式输入大小为100 * 100 * 100 得三维数组 (输入满)
3. int a[10][20]里, a[0]~a[9] 装的元素是什么呢? 如何表示
4. 若a的首地址为0, 假设a里面的每个元素都是为0 下面代码预期值应该是多少?

```

1  int a[2][2];
2  printf("%d %d %d %d %d %d %d %d",
3  a,a + 1,&a[0],&a[0] + 1,a[0],a[0] + 1,a[0][0],a[0][0] + 1,&a[0][0],&a[0][0] + 1);

```

```

1  int a[4][4];
2  printf("%d %d %d %d %d %d %d %d %d",
3  a,a + 1,&a[2],&a[2] + 1,a[2],a[2] + 1,a[1][0],a[2][1] + 1,&a[1][2],&a[3][1] + 1);

```

5. 请同学画一下, int a[2][3][4]。数组长什么样。

13.时间复杂度和空间复杂度的概念

13.1时间复杂度

在**计算机科学**中, **时间复杂性**, 又称**时间复杂度**, **算法的时间复杂度**是一个**函数**, 它定性描述该算法的运行时间。这是一个代表算法输入值的**字符串的长度**的函数。时间复杂度常用**大O符号**表述, 不包括这个函数的低阶项和首项系数。使用这种方式时, 时间复杂度可被称为是**渐近**的, 亦即考察输入值大小趋近**无穷**时的情况。


image-20221018223915662

举个例子

```

1  for(int i = 1; i <= n; i++) 就是o(n)

```

image-20221018223942665

举个例子

```

1  for(int i = 1; i <= n; i *= 2) 就是o(logn)

```

根号时间

$$T(n) = O(\sqrt{n})$$

举个例子

```
1 | for(int i = 1; i * i <= n; i++)
```

注意：计算机中一次运算算1的话，1s内大概能运行 10^8 次运算

13.2空间复杂度

空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。同样的用 $O()$ 来表示

举个例子

```
1 | int a[10000]; //他产生的空间就是4 * 10000字节
```

练习:

1. 求 n 以内所有的质数时间复杂度是多少？能不能优化？
2. 二重循环的时间复杂度是多少，三重呢？
3. 以下代码产生了多少空间大小

```
1 | int a[10000];  
2 | double b[100];  
3 | float c[101];
```

14.函数

数学中的函数就是用来代表一个数学式子的。

例如我现在有两个式子： $y = x + 1$ ， $y = x * 2$ ，我想要问你几个问题。

- ```
1 | 对于式子 $y = x + 1$ ，当 $x = 1$ 的时候答案是多少
2 | 对于式子 $y = x * 2$ ，当 $x = 1$ 的时候答案是多少
3 | 这样子非常麻烦！
```

假如我这样写： $y = f(x) = x + 1$ ， $y = T(x) = x * 2$ ，同样的

- ```
1 | f(1) 答案是多少，T(1)答案是多少，非常简单！非常好用！
```

计算机的函数，是一个固定的一个程序段，或称其为一个子程序，它在可以实现固定运算功能的同时，还带有一个入口和一个出口，所谓的入口，就是函数所带的各个参数，我们可以通过这个入口，把函数的参数值代入子程序，供计算机处理；所谓出口，就是指函数的函数值，在计算机求得之后，由此口带回给调用它的程序。像我们的main，printf其实都是一个函数，只不过C语言作者给我们做好的。所以函数就像我们的工具，我们给他做好了，以后就直接使用就好了

函数的形式是这样的的 带有（一个）参数的函数的声明：

C语言函数的分类：

1. 自定义函数

[类型名](#)标示符+函数名+（类型标示符+参数）{

// 程序代码

}

没有返回值且不带参数的函数的声明：

void+函数名（）//无类型+函数名{

// 程序代码

}

2. 库函数

像stdio.h就是一个库，printf，scanf都是库函数，后面等学到C++了我们再教大家许多库函数

14.1 函数的调用及返回

函数调用一般是这样例如

```
1 int add(int a,int b){
2     return a + b;
3 }
4 int main(){
5     add(1,2); //函数名字 + 需要传递的参数
6     return 0;
7 }
```

注意，我们的函数底层使用的是数据结构中的栈实现，每次函数调用其实就是一个压入栈的过程，当函数运行结束后，会返回到上一个函数中(主调函数),这个过程我们称之为弹栈。比如说函数A调用函数B，那么程序的控制权由函数 A 传递给函数 B,当函数 B 运行结束后，会从 函数 B 返回到函数 A,下面演示了这个过程。

调用过程

```
1 | 操作系统 -> 主函数main() -> A() -> B() -> C() -> D()
```

返回过程

```
1 | 操作系统 <- 主函数main() <- A() <- B() <- C() <- D()
```

一句话总结就是,函数从哪儿调用，就会返回到哪儿。(从哪儿来，回哪儿去)，不过值得注意的是:当函数返回的时候，可以携带结果返回，返回给主调函数。

请看下面这个例子。

自定义函数传递的参数又分为两种：

1. 形参

自定义函数中的“[形参](#)”全称为“形式参数” 由于它不是实际存在变量，所以又称[虚拟变量](#)。[实参](#)和形参可以重名。

大家可以这么理解，由于我们只告诉了函数，我们当前变量的值，所以他只能拷贝一份值去使用，所以是虚拟的，所以称作传值

```
1 就像
2  int b = 1;
3  int a = b;
4  我们创建了一个变量a去接收b的值，但是我们改变a的值，b的值是不会改变的
```

```

1  int add(int x,int y){
2      x = x + y;
3      return x;
4  }
5  int a = 1, b = 1;
6      add(a,b);
7      printf("%d",a);
8      //输出1

```

2. 实参

实际参数简称“实参”。在调用有参函数时，函数名后面括号中的参数称为“实际参数”，实参可以是常量、变量或表达式。

大家可以这么理解，相当于给当前变量取了个别名，其他都是一模一样

```

1  int a = 1;
2  int &b = a;
3  b = 2;
4  printf("%d",a);
5  //发现a输出了2

```

```

1  int add(int &x,int y){
2      x = x + y;
3      return x;
4  }
5
6  int a = 1, b = 1;
7  add(a,b);
8  printf("%d",a);
9  //输出2

```

练习:

1. 以下代码能否达到预期?

```

1  void swap(int x,int y){
2      int s = x;
3      x = y;
4      y = s;
5  }
6  int main(){
7      int a = 1,b = 2;
8      swap(a,b);
9      //能成功交换a,b的值吗?
10 }

```

```

1  void swap(int &x,int &y){
2      int s = x;
3      x = y;
4      y = s;
5  }
6  int main(){
7      int a = 1,b = 2;
8      swap(a,b);
9      //能成功交换a,b的值吗?
10 }

```

```

1 | int cheng(int x,int y){
2 |     return x * y;
3 | }
4 | int main(){
5 |     int a = 1,b = 100;
6 |     a = cheng(1,100);
7 |     //能成功获得值100吗?
8 | }

```

```

1 | int jia(int &x,int &y){
2 |     return x + y;
3 | }
4 | int cheng(int x,int y){
5 |     return x * jia(x + y);
6 | }
7 | int main(){
8 |     int a = 1,b = 100;
9 |     a = cheng(1,100);
10 |    //能成功获得值101吗?
11 | }

```

2.

```

1 | int a = 1;
2 | int &b = a;
3 | int &c = b;
4 | c = 2;
5 | //请问a的值会改变吗?

```

```

1 | int a[10] = {1,2,3};
2 | int &c = a[0];
3 | c = 12;
4 | //请问a[0]的值会改变吗?

```

```

1 | int a[10] = {1,2,3};
2 | int (&c)[10] = a;
3 | c[0] = 10;
4 | printf("%d",a[0]);
5 | //请问合法吗? 如果合法输出的值是什么

```

```

1 | int a[10] = {1,2,3};
2 | int &(c[10]) = a;
3 | c[0] = 10;
4 | printf("%d",a[0]);
5 | //请问合法吗? 如果合法输出的值是什么

```

14.2函数的声明和定义

函数的声明

类似于我们人的目录，告诉我們有哪些东西。

1. 告诉编译器有一个函数叫什么，参数是什么，返回类型是什么。但是具体是不是存在，函数声明决定不了
2. 函数的声明一般出现在函数的使用之前。要满足**先声明后使用**。
3. 函数的声明一般要放在头文件中的

函数的定义

函数的定义是指函数的具体实现，交待函数的功能实现。

练习：

1. 以下代码能否达到预期？

```
1  #include<stdio.h>
2
3  void jia(int x,int y){
4      return cheng(1,2);
5  }
6  void cheng(int x,int y){
7      return x * y;
8  }
```

```
1  #include<stdio.h>
2  void jia(int x,int y);
3  void cheng(int x,int y);
4
5  void jia(int x,int y){
6      return cheng(1,2);
7  }
8  void cheng(int x,int y){
9      return x * y;
10 }
```

```
1  #include<stdio.h>
2  int jia(int x,int y);
3  void jia(int x,int y);
```

```
1  #include<stdio.h>
2  int jia(int x,int y,int z);
3  int jia(int x,int y);
```

```
1  #include<stdio.h>
2  int jia(int x,int y);
3  int jia(int x,char y);
```

```
1  #include<stdio.h>
2  int a;
3  int jia(int a,int b){
4      printf("%d",a);
5  }
```

2. 请问函数里面的变量作用域和生命周期是？

3. 函数的作用域和生命周期是？

总结：

1. 函数是一个把行为封装起来的行为
2. 函数调用其他函数后，会保存原来的状态，直到返回。
3. 函数定义和函数声明最好分开写。
4. 函数名字可以取相同的名字，但是传递的变量需要有所不同，要么个数不同，要么变量类型不同，但是返回类型不同不行。
5. 函数可以把我们的代码很好的模块化，更简洁，更好看

15.函数递归

15.1什么是递归？

程序调用自身的编程技巧称为递归（recursion）。

举个例子

▼ 和尚讲故事

从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？“从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？‘从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？……’”

递归做为一种算法在程序设计语言中广泛应用。 一个过程或函数在其定义或说明中有直接或间接调用自身的

一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，

递归策略

只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。

递归的主要思考方式在于：把大事化小

```
1 //最简单的递归函数
2 #include<stdio.h>
3
4 int main(){
5
6     printf("hello\n");
7     main();
8     return 0;
9 }
```

15.2递归的必要条件

- 存在限制条件，当满足这个限制条件的时候，递归便不再继续。
- 每次递归调用之后越来越接近这个限制条件

练习：

1. 接受一个整型值（无符号），按照顺序打印它的每一位。

例如：

输入：1234，输出 1 2 3 4

2. 求n的阶乘

3. 求第n个斐波那契数。（不考虑溢出） $f[1] = 1, f[2] = 1, f[n] = f[n - 1] + f[n - 2]$

15.3递归和循环对比

- 递归速度慢,消耗内存高,循环速度快,消耗内存低
- 无限递归会导致系统崩溃,而无限循环会消耗 CPU 周期
- 递归使代码更小巧,而迭代使代码更长。

16.结构体

有时候要大量存储批量数据,比如说某位考生的信息,可以考虑使用数组,但是数组只能存储一组同样数据类型的信息,如果同时记录考生的信息,成绩等不同的信息就不能使用一个数组来存储了。字符串可以用来存储很多种类信息,但是如果存储数字的话,我们要一个个分隔开也很麻烦,所以这里介绍一个方法叫做结构体。C语言的结构体是由一系列具有相同类型或不同类型的数据构成的数据集合。

```
1 struct 名字1{ //大括号里面包含数据类型
2     数据类型1 成员变量1
3     数据类型2 成员变量2
4 }[结构体变量名];
5
6 struct 名字1 名字2;
```

假设我现在需要存储学生的名字,数学,语文,英语成绩,我可以这么写

```
1 struct Student{
2     char Name[10];
3     int Math;
4     int Chinese;
5     int English;
6 };
7 Student student;
8 //想要使用学生的名字就是 student.name;
9 //想要使用学生的数学成绩就是 student.math
```

现在我有 两名学生 :

Steven 他的数学语文英语成绩分别为, 100 , 90, 80。

Hellen 他的数学语文英语成绩分别为, 80 , 95, 90。

Haohao 他的数学语文英语成绩分别为, 95 , 90, 93。

```
1 //我可以这么存
2 struct Student{
3     char Name[10];
4     int Math;
5     int Chinese;
6     int English;
7 }student[2];
8 student[0] = {"Steven",100,90,80};
9 student[1].Name = "Hellen";
10 student[1].Math = 80;
11 student[1].Chinese = 80;
12 student[1].English = 80;
13 student[2] = Student{"Haohao",95,90,93};
```

17.指针

17.1 普通指针

指针，是C语言中的一个重要概念及其特点，也是掌握C语言比较困难的部分。指针也就是内存地址，指针变量是用来存放内存地址的变量，在同一CPU构架下，不同类型的指针变量所占用的存储单元长度是相同的，而存放数据的变量因数据的类型不同，所占用的存储空间长度也不同。有了指针以后，不仅可以对数据本身，也可以对存储数据的变量地址进行操作。

指针描述了数据在内存中的位置，标示了一个占据存储空间的实体，在这一段空间起始位置的相对距离值。在C/C++语言中，指针一般被认为是指针变量，指针变量的内容存储的是其指向的对象的首地址，指向的对象可以是变量（指针变量也是变量），数组，函数等占据存储空间的实体。

众所周知 &a可以是a的地址那么让我们试验一下

```
1 int b = 1;
2 int a = &b; //这样子可以写吗？
```

我们发现是过不了编译的，因为我们的内存地址只能用我们的指针来存储

```
1 int a = 1;
2 int *p = &a; //我们念作p是一个指向整形类型的指针，指向的是整形变量a的地址。
3 printf("%d %d",p,*p); //如果直接输出p的话就是输出a的地址，*p能输出该地址存储的内容，*也称作解地址符号
```

当我们更改指针的值得时候我们同时也能够更改指向那个地址的值

```
1 int a = 1;
2 int *p = &a;
3 *p = 3;
4 printf("%d",a);
```

当然我们的指针也有地址，所以我们也可以使用一个指针去指向我们的指针。

```
1 int a = 1;
2 int *p = &a;
3 int **pp = &p; //因为我们的p指向的就是a的地址，所以pp也可以直接指向
4 printf("%d",*pp); //输出1
```

注意：以下写法

```
1 int a = 1;
2 int *p = &a;
3 int **pp = &p; //他这样叫，pp是一个指针指向的是一个指向整形变量地址的指针的地址，所以他必须指向一个指针的地址。
4 //不能这么写 int **pp = p;
5 printf("%d",**pp); //输出1
```

注意：指针不能直接指向一块地址例如：

```
1 int *p = 1;
2 //这是禁止的，因为我们的空间也会保存我们操作系统的一些东西，如果我们直接能够使用地址去更改的话，会导致操作系统出错
3 //所以在操作指针的时候需要注意
```

练习:

1. 以下操作合法吗?若合法输出的结果是什么?

```
1 int a[100];
2 a[0] = 1;
3 int *p = a;
4 printf("%d", *p);
```

```
1 int a[100];
2 a[0] = 1, a[1] = 2;
3 int *p = a;
4 printf("%d", *(p + 1));
```

```
1 int a = 10;
2 int *p = &a;
3 int **pp = &p;
4 int ***ppp = &pp;
5 printf("%d", ***ppp);
```

```
1 //倘若a的地址为0 *p的地址为 10, **pp的地址为100, ***ppp的地址为1000
2 int a = 10;
3 int *p = &a;
4 int **pp = &p;
5 int ***ppp = &pp;
6 printf("%d", ***ppp);
7 printf("%d", **ppp);
8 printf("%d", *ppp);
9 printf("%d", ppp);
10 printf("%d\n", &ppp);
11 printf("%d\n", &pp);
12 printf("%d\n", &p);
13 printf("%d\n", &a);
```

```
1 int a = 100;
2 int *p = &a;
3 int **pp = &p;
4 int *c = &pp;
5 printf("%d", *c);
```

17.2数组指针，指针数组

大家看名字可能看不出什么区别，但是在他们名字中间加个的就很明确了

数组的指针

说明他是一个指针，什么指针，指向数组

```
1 int (*p)[10]; //这个就读作p是一个指针指向一个大小为10的数组的首地址
2 int a[10] = {1};
3 p = &a;
4 printf("%d", (*p)[0]);
```


指针的数组

说明他是一个数组，什么数组，里面都是指针的数组

```
1 int *(p[10]);
2 int a = 1;
3 p[0] = &a;
4 printf("%d",*(p[0]));
```

那么如果我们不写括号的话是什么呢？

```
1 int *p[10]; //他先和数组结合，所以他是指针数组
```

18.C语言中其他关键词

18.1 define宏定义

宏是在程序运行前的一个准备工作

无参宏

一种最简单的宏的形式如下：

```
1 一种最简单的宏的形式如下：
2  #define 宏名 替换文本
   每个#define行（即逻辑行）由三部分组成：第一部分是指令 #define 自身，“#”表示这是一条预处理命令，“define”为宏命令。第二部分为宏（macro），一般为缩略语，其名称（宏名）一般大写，而且不能有空格，遵循C变量命名规则。“替换文本”可以是任意常数、表达式、字符串等。在预处理工作过程中，代码中所有出现的“宏名”，都会被“替换文本”替换。这个替换的过程被称为“宏代换”或“宏展开”（macro expansion）。“宏代换”是由预处理程序自动完成的。在C语言中，“宏”分为两种：无参数 和 有参数。
```

无参宏是指宏名之后不带参数，上面最简单的宏就是无参宏。

```
1 #define M 5 // 宏定义
2 #define PI 3.14 //宏定义
3 int a[M]; // 会被替换为: int a[5];
4 int b = M; // 会被替换为: int b = 5;
5 printf("PI = %.2f\n", PI); // 输出结果为: PI = 3.14
```

注意宏不是语句，结尾不需要加“;”，否则会被替换进程序中，如：

```
1 #define N 10; // 宏定义
2 int c[N]; // 会被替换为: int c[10];
3 //error:... main.c:133:11: Expected ']'
```

如果要写宏不止一行，则在结尾加反斜线符号使得多行能连接上，如：

```
1 #define HELLO "hello \
2 the world"
```

注意第二行要对齐，否则，如：

```
1 #define HELLO "hello the wo\
2 rld"
3 printf("HELLO is %s\n", HELLO);
4 //输出结果为: HELLO is hello the wo rld
```

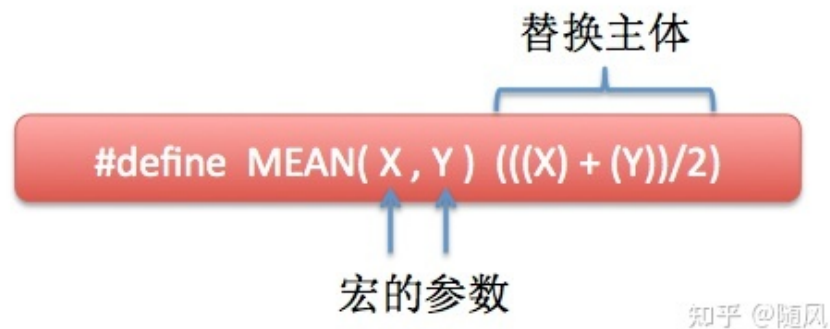
也就是行与行之间的空格也会被作为替换文本的一部分

而且由这个例子也可以看出：宏名如果出现在源程序中的“”内，则不会被当做宏来进行宏代换。

宏可以嵌套，但不参与运算：

```
1 #define M 5 // 宏定义
2 #define MM M * M // 宏的嵌套
3 printf("MM = %d\n", MM); // MM 被替换为：MM = M * M，然后又变成 MM = 5 * 5
```

有参宏



宏调用：

宏名（实参表）；

```
printf("MEAN = %d\n", MEAN(7, 9)); // 输出结果： MEAN = 8
```

和函数类似，在宏定义中的参数成为形式参数，在宏调用中的参数成为实际参数。

而且和无参宏不同的一点是，有参宏在调用中，不仅要进行宏展开，而且还要用实参去替换形参。如：

```
1 #define M 5 //无参宏
2 #define COUNT(M) M * M //有参宏
3 printf("COUNT = %d\n", COUNT(10)); // 替换为： COUNT(10) = 10 * 10
4 // 输出结果： COUNT = 100
```

这看上去用法与函数调用类似，但实际上是有很大差别的。如：

```
1 #define COUNT(M) M * M //定义有参宏
2 int x = 6;
3 printf("COUNT = %d\n", COUNT(x + 1)); // 输出结果： COUNT = 13
4 printf("COUNT = %d\n", COUNT(++x)); // 输出结果： COUNT = 56
```

这两个结果和调用函数的方法的结果差别很大，因为如果是像函数那样的话，COUNT(x + 1)应该相当于COUNT(7)，结果应该是 $7 * 7 = 49$ ，但输出结果却是21。原因在于，预处理器不进行技术，只是进行字符串替换，而且也不会自动加上括号（），所以COUNT(x + 1)被替换为COUNT(x + 1 * x + 1)，代入x = 6，即为 $6 + 1 * 6 + 1 = 13$ 。而解决办法则是：尽量用括号把整个替换文本及其中的每个参数括起来：

```
1 #define COUNT(M) ((M) * (M))
```

但即使用括号，也不能解决上面例子的最后一个情况，COUNT(++x) 被替换为 ++x * ++x，即为 $7 * 8 = 56$ ，而不是想要 $7 * 7 = 49$ ，解决办法最简单的是：不要在有参宏用使用到“++”、“--”等。

练习:

1. 以下代码输出什么?

```
1 | #define x 5 + 1
2 | printf("%d",x * x);
```

```
1 | #define x (5 + 1)
2 | printf("%d",x * x);
```

```
1 | #define add(x,y) x + y
2 | printf("%d\n",add(1,2));
```

18.2 typedef

typedef是在[计算机编程语言](#)中用来为复杂的声明定义简单的别名，它与宏定义有些差异。它本身是一种存储类的关键字，与auto、extern、mutable、static、register等关键字不能出现在同一个表达式中。

我们一般用来给我们的变量取别名例如：

```
1 | typedef long long ll;
2 | typedef double db;
3 |
4 | ll a;db b;
```

当然你也可以使用宏定义

```
1 | #define ll long long
2 | #define db double
```

18.3 const

const是一个C语言（ANSI C）的关键字，具有着举足轻重的地位。它限定一个变量不允许被改变，产生静态作用。使用const在一定程度上可以提高程序的安全性和可靠性。另外，在观看别人代码的时候，清晰理解const所起的作用，对理解对方的程序也有一定帮助

因为我们的变量都是可变的，所以也可以有常量，不能更改的变量
使用方法

```
1 | const int a = 1000;
```

但是以下方法不行

```
1 | const int a;
2 | a = 1000;//后面赋值就错误
```

常量常常用来定义全局变量

以下写法编译是过不了的，因为全局变量的数组大小只能用常量定义

```
1 | #include<stdio.h>
2 | int a = 100;
3 | int b[a];
4 | int main(){
5 |     return 0;
6 | }
```

以下正确

```
1 | #include<stdio.h>
2 | const int a = 100;
3 | int b[a];
4 | int main(){
5 |     return 0;
6 | }
```

当然常量也可以用来定义指针

```
1 | int a = 10;
2 | const int *p = &a;
```

但是大家觉得以下的代码正确吗？

```
1 | int a = 10;
2 | const int *p = &a;
3 | a = 100;
4 | printf("%d",*p);
```

我们发现编译成功了，并且*p的值输出了100

但是我们直接修改*p是不可以的，所以指针常量的值还是可能会改变的

还有一个要注意的点:你们觉得以下代码对吗？

```
1 | int a = 1,b = 2;
2 | const int *p = &a;
3 | p = &b;
4 | printf("%d\n",*p);
```

我们发现是可以编译的，并且成功输出了2

所以我们发现 const int *p; 仅仅是不能改变 *p的值，但是p指向的地址可以改变。

所以我们想要 *p和地址都不改变我们需要这样子写

```
1 | int a = 1,b = 2;
2 | const int * const p = &a;
3 | p = &b;
4 | printf("%d\n",*p);
```